Problem 1764: Form Array by Concatenating Subarrays of Another Array

Problem Information

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

Problem Description

You are given a 2D integer array

groups

of length

n

. You are also given an integer array

nums

.

You are asked if you can choose

n

disjoint

subarrays from the array

nums

such that the

i

th

subarray is equal to

groups[i]

(

0-indexed

), and if

i > 0

, the

(i-1)

th

subarray appears

before

the

i

th

subarray in

nums

(i.e. the subarrays must be in the same order as

groups

).

Return

true

if you can do this task, and

false

otherwise

.

Note that the subarrays are

disjoint

if and only if there is no index

k

such that

nums[k]

belongs to more than one subarray. A subarray is a contiguous sequence of elements within an array.

Example 1:

Input:

groups = [[1,-1,-1],[3,-2,0]], nums = [1,-1,0,1,-1,-1,3,-2,0]

Output:

true

Explanation:

You can choose the 0

th

subarray as [1,-1,0,

1,-1,-1

,3,-2,0] and the 1

st

one as [1,-1,0,1,-1,-1,

3,-2,0

]. These subarrays are disjoint as they share no common nums[k] element.

Example 2:

Input:

groups = [[10,-2],[1,2,3,4]], nums = [1,2,3,4,10,-2]

Output:

false

Explanation:

Note that choosing the subarrays [

1,2,3,4

,10,-2] and [1,2,3,4,

10,-2

] is incorrect because they are not in the same order as in groups. [10,-2] must come before [1,2,3,4].

Example 3:

Input:

groups = [[1,2,3],[3,4]], nums = [7,7,1,2,3,4,7,7]

Output:

false

Explanation:

Note that choosing the subarrays [7,7,

1,2,3

,4,7,7] and [7,7,1,2,

3,4

,7,7] is invalid because they are not disjoint. They share a common elements nums[4] (0-indexed).

Constraints:

groups.length == n

1 <= n <= 10

3

1 <= groups[i].length, sum(groups[i].length) <= 10

3

1 <= nums.length <= 10

3

-10

7

<= groups[i][j], nums[k] <= 10

7

Code Snippets

C++:

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

}
};
```

Java:

```
class Solution {
public boolean canChoose(int[][] groups, int[] nums) {

}
}
```

Python3:

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

Python:

```
class Solution(object):
def canChoose(self, groups, nums):
```

```
"""
:type groups: List[List[int]]
:type nums: List[int]
:rtype: bool
"""
```

JavaScript:

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

};
```

TypeScript:

```
function canChoose(groups: number[][], nums: number[]): boolean {

};
```

C#:

```
public class Solution {
public bool CanChoose(int[][] groups, int[] nums) {

}
}
```

C:

```
bool canChoose(int** groups, int groupsSize, int* groupsColSize, int* nums,
int numsSize) {

}
```

Go:

```
func canChoose(groups [][]int, nums []int) bool {
```

```
}
```

Kotlin:

```
class Solution {
fun canChoose(groups: Array<IntArray>, nums: IntArray): Boolean {

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

/**
* @param Integer[][] $groups
* @param Integer[] $nums
```

```
* @return Boolean
*/
function canChoose($groups, $nums) {

}
}
```

Dart:

```
class Solution {
bool canChoose(List<List<int>> groups, List<int> nums) {

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

```
-spec can_choose(Groups :: [[integer()]], Nums :: [integer()]) -> boolean().
can_choose(Groups, Nums) ->
.
```

Racket:

```
(define/contract (can-choose groups nums)
(-> (listof (listof exact-integer?)) (listof exact-integer?) boolean?)
```

```
)
```

Solutions

C++ Solution:

```
/*
* Problem: Form Array by Concatenating Subarrays of Another Array
* Difficulty: Medium
* Tags: array, string, greedy
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

class Solution {
public:
bool canChoose(vector<vector<int>>& groups, vector<int>& nums) {

}
};
```

Java Solution:

```
/**
* Problem: Form Array by Concatenating Subarrays of Another Array
* Difficulty: Medium
* Tags: array, string, greedy
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

class Solution {
public boolean canChoose(int[][] groups, int[] nums) {

}
}
```

Python3 Solution:

```
"""
Problem: Form Array by Concatenating Subarrays of Another Array
Difficulty: Medium
Tags: array, string, greedy

Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(1) to O(n) depending on approach
"""

class Solution:
def canChoose(self, groups: List[List[int]], nums: List[int]) -> bool:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Form Array by Concatenating Subarrays of Another Array
* Difficulty: Medium
* Tags: array, string, greedy
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

/**
* @param {number[][]} groups
* @param {number[]} nums
```

```
 * @return {boolean}
 */
var canChoose = function(groups, nums) {

};
```

TypeScript Solution:

```
/**
* Problem: Form Array by Concatenating Subarrays of Another Array
* Difficulty: Medium
* Tags: array, string, greedy
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

function canChoose(groups: number[][], nums: number[]): boolean {

};
```

C# Solution:

```
/*
* Problem: Form Array by Concatenating Subarrays of Another Array
* Difficulty: Medium
* Tags: array, string, greedy
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

public class Solution {
public bool CanChoose(int[][] groups, int[] nums) {

}
}
```

C Solution:

```
/*
* Problem: Form Array by Concatenating Subarrays of Another Array
* Difficulty: Medium
* Tags: array, string, greedy
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

bool canChoose(int** groups, int groupsSize, int* groupsColSize, int* nums,
int numsSize) {

}
```

Go Solution:

```
// Problem: Form Array by Concatenating Subarrays of Another Array
// Difficulty: Medium
// Tags: array, string, greedy
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(1) to O(n) depending on approach

func canChoose(groups [][]int, nums []int) bool {

}
```

Kotlin Solution:

```
class Solution {
fun canChoose(groups: Array<IntArray>, nums: IntArray): Boolean {

}
}
```

Swift Solution:

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

```
}
}
```

Rust Solution:

```
// Problem: Form Array by Concatenating Subarrays of Another Array
// Difficulty: Medium
// Tags: array, string, greedy
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(1) to O(n) depending on approach

impl Solution {
pub fn can_choose(groups: Vec<Vec<i32>>, nums: Vec<i32>) -> bool {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

```
class Solution {
bool canChoose(List<List<int>> groups, List<int> nums) {

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

```
-spec can_choose(Groups :: [[integer()]], Nums :: [integer()]) -> boolean().
can_choose(Groups, Nums) ->
.
```

Racket Solution:

```
(define/contract (can-choose groups nums)
(-> (listof (listof exact-integer?)) (listof exact-integer?) boolean?)
)
```