

Problem 3447: Assign Elements to Groups with Constraints

Problem Information

Difficulty: Medium

Acceptance Rate: 0.00%

Paid Only: No

Problem Description

You are given an integer array

groups

, where

groups[i]

represents the size of the

i

th

group. You are also given an integer array

elements

.

Your task is to assign

one

element to each group based on the following rules:

An element at index

j

can be assigned to a group

i

if

groups[i]

is

divisible

by

elements[j]

If there are multiple elements that can be assigned, assign the element with the

smallest index

j

If no element satisfies the condition for a group, assign -1 to that group.

Return an integer array

assigned

, where

`assigned[i]`

is the index of the element chosen for group

`i`

, or -1 if no suitable element exists.

Note

: An element may be assigned to more than one group.

Example 1:

Input:

`groups = [8,4,3,2,4], elements = [4,2]`

Output:

`[0,0,-1,1,0]`

Explanation:

`elements[0] = 4`

is assigned to groups 0, 1, and 4.

`elements[1] = 2`

is assigned to group 3.

Group 2 cannot be assigned any element.

Example 2:

Input:

`groups = [2,3,5,7], elements = [5,3,3]`

Output:

`[-1,1,0,-1]`

Explanation:

`elements[1] = 3`

is assigned to group 1.

`elements[0] = 5`

is assigned to group 2.

Groups 0 and 3 cannot be assigned any element.

Example 3:

Input:

`groups = [10,21,30,41], elements = [2,1]`

Output:

`[0,1,0,1]`

Explanation:

`elements[0] = 2`

is assigned to the groups with even values, and

`elements[1] = 1`

is assigned to the groups with odd values.

Constraints:

$1 \leq \text{groups.length} \leq 10$

5

$1 \leq \text{elements.length} \leq 10$

5

$1 \leq \text{groups}[i] \leq 10$

5

$1 \leq \text{elements}[i] \leq 10$

5

Code Snippets

C++:

```
class Solution {
public:
vector<int> assignElements(vector<int>& groups, vector<int>& elements) {
    }
};
```

Java:

```
class Solution {
public int[] assignElements(int[] groups, int[] elements) {
    }
}
```

Python3:

```
class Solution:
def assignElements(self, groups: List[int], elements: List[int]) ->
List[int]:
```

Python:

```
class Solution(object):
    def assignElements(self, groups, elements):
        """
        :type groups: List[int]
        :type elements: List[int]
        :rtype: List[int]
        """
```

JavaScript:

```
/**
 * @param {number[]} groups
 * @param {number[]} elements
 * @return {number[]}
 */
var assignElements = function(groups, elements) {
}
```

TypeScript:

```
function assignElements(groups: number[], elements: number[]): number[] {
}
```

C#:

```
public class Solution {
    public int[] AssignElements(int[] groups, int[] elements) {
    }
}
```

C:

```
/**
 * Note: The returned array must be malloced, assume caller calls free().
 */
int* assignElements(int* groups, int groupsSize, int* elements, int
elementsSize, int* returnSize) {
```

```
}
```

Go:

```
func assignElements(groups []int, elements []int) []int {  
    }  
}
```

Kotlin:

```
class Solution {  
    fun assignElements(groups: IntArray, elements: IntArray): IntArray {  
        }  
        }  
}
```

Swift:

```
class Solution {  
    func assignElements(_ groups: [Int], _ elements: [Int]) -> [Int] {  
        }  
        }  
}
```

Rust:

```
impl Solution {  
    pub fn assign_elements(groups: Vec<i32>, elements: Vec<i32>) -> Vec<i32> {  
        }  
        }  
}
```

Ruby:

```
# @param {Integer[]} groups  
# @param {Integer[]} elements  
# @return {Integer[]}  
def assign_elements(groups, elements)  
  
end
```

PHP:

```

class Solution {

    /**
     * @param Integer[] $groups
     * @param Integer[] $elements
     * @return Integer[]
     */
    function assignElements($groups, $elements) {

    }
}

```

Dart:

```

class Solution {
List<int> assignElements(List<int> groups, List<int> elements) {
    }
}

```

Scala:

```

object Solution {
def assignElements(groups: Array[Int], elements: Array[Int]): Array[Int] = {
    }
}

```

Elixir:

```

defmodule Solution do
@spec assign_elements(groups :: [integer], elements :: [integer]) :: [integer]
def assign_elements(groups, elements) do
    end
end

```

Erlang:

```

-spec assign_elements(Groups :: [integer()], Elements :: [integer()]) ->
[integer()].
assign_elements(Groups, Elements) ->

```

.

Racket:

```
(define/contract (assign-elements groups elements)
  (-> (listof exact-integer?) (listof exact-integer?) (listof exact-integer?))
    )
```

Solutions

C++ Solution:

```
/*
 * Problem: Assign Elements to Groups with Constraints
 * 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:
vector<int> assignElements(vector<int>& groups, vector<int>& elements) {

}
};
```

Java Solution:

```
/**
 * Problem: Assign Elements to Groups with Constraints
 * 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 int[] assignElements(int[] groups, int[] elements) {  
  
    }  
}
```

Python3 Solution:

```
"""  
  
Problem: Assign Elements to Groups with Constraints  
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 assignElements(self, groups: List[int], elements: List[int]) ->  
        List[int]:  
        # TODO: Implement optimized solution  
        pass
```

Python Solution:

```
class Solution(object):  
    def assignElements(self, groups, elements):  
        """  
        :type groups: List[int]  
        :type elements: List[int]  
        :rtype: List[int]  
        """
```

JavaScript Solution:

```
/**  
 * Problem: Assign Elements to Groups with Constraints  
 * 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[]} groups
 * @param {number[]} elements
 * @return {number[]}
 */
var assignElements = function(groups, elements) {

};

```

TypeScript Solution:

```

/**
 * Problem: Assign Elements to Groups with Constraints
 * 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 assignElements(groups: number[], elements: number[]): number[] {

};

```

C# Solution:

```

/*
 * Problem: Assign Elements to Groups with Constraints
 * 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 int[] AssignElements(int[] groups, int[] elements) {
}
}

```

C Solution:

```

/*
 * Problem: Assign Elements to Groups with Constraints
 * 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
 */

/**
 * Note: The returned array must be malloced, assume caller calls free().
 */
int* assignElements(int* groups, int groupsSize, int* elements, int
elementsSize, int* returnSize) {

}

```

Go Solution:

```

// Problem: Assign Elements to Groups with Constraints
// 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 assignElements(groups []int, elements []int) []int {
}

```

Kotlin Solution:

```
class Solution {  
    fun assignElements(groups: IntArray, elements: IntArray): IntArray {  
        //  
        //  
    }  
}
```

Swift Solution:

```
class Solution {  
    func assignElements(_ groups: [Int], _ elements: [Int]) -> [Int] {  
        //  
        //  
    }  
}
```

Rust Solution:

```
// Problem: Assign Elements to Groups with Constraints  
// 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 assign_elements(groups: Vec<i32>, elements: Vec<i32>) -> Vec<i32> {  
        //  
        //  
    }  
}
```

Ruby Solution:

```
# @param {Integer[]} groups  
# @param {Integer[]} elements  
# @return {Integer[]}  
def assign_elements(groups, elements)  
  
end
```

PHP Solution:

```
class Solution {  
  
    /**  
     * @param Integer[] $groups  
     * @param Integer[] $elements  
     * @return Integer[]  
     */  
    function assignElements($groups, $elements) {  
  
    }  
}
```

Dart Solution:

```
class Solution {  
List<int> assignElements(List<int> groups, List<int> elements) {  
  
}  
}
```

Scala Solution:

```
object Solution {  
def assignElements(groups: Array[Int], elements: Array[Int]): Array[Int] = {  
  
}  
}
```

Elixir Solution:

```
defmodule Solution do  
@spec assign_elements([integer], [integer]) :: [integer]  
def assign_elements(groups, elements) do  
  
end  
end
```

Erlang Solution:

```
-spec assign_elements(Groups :: [integer()], Elements :: [integer()]) ->
[integer()].
assign_elements(Groups, Elements) ->
.
```

Racket Solution:

```
(define/contract (assign-elements groups elements)
  (-> (listof exact-integer?) (listof exact-integer?) (listof exact-integer?))
)
```