Problem 1282: Group the People Given the Group Size They Belong To

Problem Information

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

Problem Description

There are

n

people that are split into some unknown number of groups. Each person is labeled with a

unique ID

from

0

to

n - 1

.

You are given an integer array

groupSizes

, where

groupSizes[i]

is the size of the group that person

i

is in. For example, if

groupSizes[1] = 3

, then person

1

must be in a group of size

3

.

Return

a list of groups such that each person

i

is in a group of size

groupSizes[i]

.

Each person should appear in

exactly one group

, and every person must be in a group. If there are multiple answers,

return any of them

. It is

guaranteed

that there will be

at least one

valid solution for the given input.

Example 1:

Input:

groupSizes = [3,3,3,3,3,1,3]

Output:

[[5],[0,1,2],[3,4,6]]

Explanation:

The first group is [5]. The size is 1, and groupSizes[5] = 1. The second group is [0,1,2]. The size is 3, and groupSizes[0] = groupSizes[1] = groupSizes[2] = 3. The third group is [3,4,6]. The size is 3, and groupSizes[3] = groupSizes[4] = groupSizes[6] = 3. Other possible solutions are [[2,1,6],[5],[0,4,3]] and [[5],[0,6,2],[4,3,1]].

Example 2:

Input:

groupSizes = [2,1,3,3,3,2]

Output:

[[1],[0,5],[2,3,4]]

Constraints:

groupSizes.length == n

1 <= n <= 500

1 <= groupSizes[i] <= n

Code Snippets

C++:

```
class Solution {
public:
vector<vector<int>> groupThePeople(vector<int>& groupSizes) {

}
};
```

Java:

```
class Solution {
public List<List<Integer>> groupThePeople(int[] groupSizes) {

}
}
```

Python3:

```
class Solution:
def groupThePeople(self, groupSizes: List[int]) -> List[List[int]]:
```

Python:

```
class Solution(object):
def groupThePeople(self, groupSizes):
"""
:type groupSizes: List[int]
:rtype: List[List[int]]
"""
```

JavaScript:

```
/**
* @param {number[]} groupSizes
* @return {number[][]}
*/
var groupThePeople = function(groupSizes) {

};
```

TypeScript:

```
function groupThePeople(groupSizes: number[]): number[][] {

};
```

C#:

```
public class Solution {
public IList<IList<int>> GroupThePeople(int[] groupSizes) {

}
}
```

C:

```
/**
* Return an array of arrays of size *returnSize.
* The sizes of the arrays are returned as *returnColumnSizes array.
* Note: Both returned array and *columnSizes array must be malloced, assume
caller calls free().
*/
int** groupThePeople(int* groupSizes, int groupSizesSize, int* returnSize,
int** returnColumnSizes) {

}
```

Go:

```
func groupThePeople(groupSizes []int) [][]int {

}
```

Kotlin:

```
class Solution {
fun groupThePeople(groupSizes: IntArray): List<List<Int>> {

}
}
```

Swift:

```
class Solution {
func groupThePeople(_ groupSizes: [Int]) -> [[Int]] {

}
}
```

Rust:

```
impl Solution {
pub fn group_the_people(group_sizes: Vec<i32>) -> Vec<Vec<i32>> {

}
}
```

Ruby:

```
# @param {Integer[]} group_sizes
# @return {Integer[][]}
def group_the_people(group_sizes)

end
```

PHP:

```
class Solution {

/**
* @param Integer[] $groupSizes
* @return Integer[][]
*/
function groupThePeople($groupSizes) {

}
}
```

Dart:

```
class Solution {
List<List<int>> groupThePeople(List<int> groupSizes) {

}
}
```

Scala:

```
object Solution {
def groupThePeople(groupSizes: Array[Int]): List[List[Int]] = {

}
}
```

Elixir:

```
defmodule Solution do
@spec group_the_people(group_sizes :: [integer]) :: [[integer]]
def group_the_people(group_sizes) do

end
end
```

Erlang:

```
-spec group_the_people(GroupSizes :: [integer()]) -> [[integer()]].
group_the_people(GroupSizes) ->
.
```

Racket:

```
(define/contract (group-the-people groupSizes)
(-> (listof exact-integer?) (listof (listof exact-integer?)))
)
```

Solutions

C++ Solution:

```
/*
* Problem: Group the People Given the Group Size They Belong To
* Difficulty: Medium
* Tags: array, greedy, 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<vector<int>> groupThePeople(vector<int>& groupSizes) {

}
};
```

Java Solution:

```
/**
* Problem: Group the People Given the Group Size They Belong To
* Difficulty: Medium
* Tags: array, greedy, 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 List<List<Integer>> groupThePeople(int[] groupSizes) {

}
}
```

Python3 Solution:

```
"""
Problem: Group the People Given the Group Size They Belong To
Difficulty: Medium
Tags: array, greedy, 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 groupThePeople(self, groupSizes: List[int]) -> List[List[int]]:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def groupThePeople(self, groupSizes):
"""
:type groupSizes: List[int]
:rtype: List[List[int]]
"""
```

JavaScript Solution:

```
/**
* Problem: Group the People Given the Group Size They Belong To
* Difficulty: Medium
* Tags: array, greedy, 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[]} groupSizes
* @return {number[][]}
*/
var groupThePeople = function(groupSizes) {

};
```

TypeScript Solution:

```
/**
* Problem: Group the People Given the Group Size They Belong To
* Difficulty: Medium
* Tags: array, greedy, 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 groupThePeople(groupSizes: number[]): number[][] {

};
```

C# Solution:

```
/*
* Problem: Group the People Given the Group Size They Belong To
* Difficulty: Medium
* Tags: array, greedy, 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 IList<IList<int>> GroupThePeople(int[] groupSizes) {

}
}
```

C Solution:

```
/*
* Problem: Group the People Given the Group Size They Belong To
* Difficulty: Medium
* Tags: array, greedy, 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
```

```
*/

/**
* Return an array of arrays of size *returnSize.
* The sizes of the arrays are returned as *returnColumnSizes array.
* Note: Both returned array and *columnSizes array must be malloced, assume
caller calls free().
*/
int** groupThePeople(int* groupSizes, int groupSizesSize, int* returnSize,
int** returnColumnSizes) {

}
```

Go Solution:

```
// Problem: Group the People Given the Group Size They Belong To
// Difficulty: Medium
// Tags: array, greedy, 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 groupThePeople(groupSizes []int) [][]int {

}
```

Kotlin Solution:

```
class Solution {
fun groupThePeople(groupSizes: IntArray): List<List<Int>> {

}
}
```

Swift Solution:

```
class Solution {
func groupThePeople(_ groupSizes: [Int]) -> [[Int]] {

}
```

```
}
```

Rust Solution:

```
// Problem: Group the People Given the Group Size They Belong To
// Difficulty: Medium
// Tags: array, greedy, 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 group_the_people(group_sizes: Vec<i32>) -> Vec<Vec<i32>> {

}
}
```

Ruby Solution:

```
# @param {Integer[]} group_sizes
# @return {Integer[][]}
def group_the_people(group_sizes)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[] $groupSizes
* @return Integer[][]
*/
function groupThePeople($groupSizes) {

}
}
```

Dart Solution:

```
class Solution {
List<List<int>> groupThePeople(List<int> groupSizes) {

}
}
```

Scala Solution:

```
object Solution {
def groupThePeople(groupSizes: Array[Int]): List[List[Int]] = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec group_the_people(group_sizes :: [integer]) :: [[integer]]
def group_the_people(group_sizes) do

end
end
```

Erlang Solution:

```
-spec group_the_people(GroupSizes :: [integer()]) -> [[integer()]].
group_the_people(GroupSizes) ->
.
```

Racket Solution:

```
(define/contract (group-the-people groupSizes)
(-> (listof exact-integer?) (listof (listof exact-integer?)))
)
```