Problem 851: Loud and Rich

Problem Information

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

Problem Description

There is a group of

n

people labeled from

0

to

n - 1

where each person has a different amount of money and a different level of quietness.

You are given an array

richer

where

richer[i] = [a

i

, b

i

]

indicates that

a

i

has more money than

b

i

and an integer array

quiet

where

quiet[i]

is the quietness of the

i

th

person. All the given data in richer are

logically correct

(i.e., the data will not lead you to a situation where

x

is richer than

y

and

y

is richer than

x

at the same time).

Return

an integer array

answer

where

answer[x] = y

if

y

is the least quiet person (that is, the person

y

with the smallest value of

quiet[y]

) among all people who definitely have equal to or more money than the person

x

.

Example 1:

Input:

richer = [[1,0],[2,1],[3,1],[3,7],[4,3],[5,3],[6,3]], quiet = [3,2,5,4,6,1,7,0]

Output:

[5,5,2,5,4,5,6,7]

Explanation:

answer[0] = 5. Person 5 has more money than 3, which has more money than 1, which has more money than 0. The only person who is quieter (has lower quiet[x]) is person 7, but it is not clear if they have more money than person 0. answer[7] = 7. Among all people that definitely have equal to or more money than person 7 (which could be persons 3, 4, 5, 6, or 7), the person who is the quietest (has lower quiet[x]) is person 7. The other answers can be filled out with similar reasoning.

Example 2:

Input:

richer = [], quiet = [0]

Output:

[0]

Constraints:

n == quiet.length

1 <= n <= 500

0 <= quiet[i] < n

All the values of

quiet

are

unique

.

0 <= richer.length <= n * (n - 1) / 2

0 <= a

i

, b

i

< n

a

i

!= b

i

All the pairs of

richer

are

unique

.

The observations in

richer

are all logically consistent.

Code Snippets

C++:

```
class Solution {
public:
vector<int> loudAndRich(vector<vector<int>>& richer, vector<int>& quiet) {

}
};
```

Java:

```
class Solution {
public int[] loudAndRich(int[][] richer, int[] quiet) {

}
}
```

Python3:

```
class Solution:
def loudAndRich(self, richer: List[List[int]], quiet: List[int]) ->
List[int]:
```

Python:

```
class Solution(object):
def loudAndRich(self, richer, quiet):
"""
:type richer: List[List[int]]
:type quiet: List[int]
:rtype: List[int]
"""
```

JavaScript:

```
/**
* @param {number[][]} richer
* @param {number[]} quiet
* @return {number[]}
*/
var loudAndRich = function(richer, quiet) {

};
```

TypeScript:

```
function loudAndRich(richer: number[][], quiet: number[]): number[] {

};
```

C#:

```
public class Solution {
public int[] LoudAndRich(int[][] richer, int[] quiet) {

}
}
```

C:

```
/**
* Note: The returned array must be malloced, assume caller calls free().
*/
int* loudAndRich(int** richer, int richerSize, int* richerColSize, int*
quiet, int quietSize, int* returnSize) {

}
```

Go:

```
func loudAndRich(richer [][]int, quiet []int) []int {

}
```

Kotlin:

```
class Solution {
fun loudAndRich(richer: Array<IntArray>, quiet: IntArray): IntArray {

}
}
```

Swift:

```
class Solution {
func loudAndRich(_ richer: [[Int]], _ quiet: [Int]) -> [Int] {

}
}
```

Rust:

```
impl Solution {
pub fn loud_and_rich(richer: Vec<Vec<i32>>, quiet: Vec<i32>) -> Vec<i32> {

}
}
```

Ruby:

```
# @param {Integer[][]} richer
# @param {Integer[]} quiet
# @return {Integer[]}
def loud_and_rich(richer, quiet)

end
```

PHP:

```
class Solution {

/**
* @param Integer[][] $richer
* @param Integer[] $quiet
* @return Integer[]
*/
function loudAndRich($richer, $quiet) {

}
}
```

Dart:

```
class Solution {
List<int> loudAndRich(List<List<int>> richer, List<int> quiet) {

}
}
```

Scala:

```
object Solution {
def loudAndRich(richer: Array[Array[Int]], quiet: Array[Int]): Array[Int] = {

}
}
```

Elixir:

```
defmodule Solution do
@spec loud_and_rich(richer :: [[integer]], quiet :: [integer]) :: [integer]
def loud_and_rich(richer, quiet) do

end
end
```

Erlang:

```
-spec loud_and_rich(Richer :: [[integer()]], Quiet :: [integer()]) ->
[integer()].
loud_and_rich(Richer, Quiet) ->
.
```

Racket:

```
(define/contract (loud-and-rich richer quiet)
(-> (listof (listof exact-integer?)) (listof exact-integer?) (listof
exact-integer?))
)
```

Solutions

C++ Solution:

```
/*
* Problem: Loud and Rich
* Difficulty: Medium
* Tags: array, graph, sort, search
*
* 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:
vector<int> loudAndRich(vector<vector<int>>& richer, vector<int>& quiet) {

}
};
```

Java Solution:

```
/**
* Problem: Loud and Rich
* Difficulty: Medium
* Tags: array, graph, sort, search
*
* 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 int[] loudAndRich(int[][] richer, int[] quiet) {

}
}
```

Python3 Solution:

```
"""
Problem: Loud and Rich
```

```
Difficulty: Medium
Tags: array, graph, sort, search

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 loudAndRich(self, richer: List[List[int]], quiet: List[int]) ->
List[int]:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def loudAndRich(self, richer, quiet):
"""
:type richer: List[List[int]]
:type quiet: List[int]
:rtype: List[int]
"""
```

JavaScript Solution:

```
/**
* Problem: Loud and Rich
* Difficulty: Medium
* Tags: array, graph, sort, search
*
* 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[][]} richer
* @param {number[]} quiet
* @return {number[]}
*/
```

```
var loudAndRich = function(richer, quiet) {

};
```

TypeScript Solution:

```
/**
* Problem: Loud and Rich
* Difficulty: Medium
* Tags: array, graph, sort, search
*
* 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 loudAndRich(richer: number[][], quiet: number[]): number[] {

};
```

C# Solution:

```
/*
* Problem: Loud and Rich
* Difficulty: Medium
* Tags: array, graph, sort, search
*
* 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 int[] LoudAndRich(int[][] richer, int[] quiet) {

}
}
```

C Solution:

```
/*
* Problem: Loud and Rich
* Difficulty: Medium
* Tags: array, graph, sort, search
*
* 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
*/

/**
* Note: The returned array must be malloced, assume caller calls free().
*/
int* loudAndRich(int** richer, int richerSize, int* richerColSize, int*
quiet, int quietSize, int* returnSize) {

}
```

Go Solution:

```
// Problem: Loud and Rich
// Difficulty: Medium
// Tags: array, graph, sort, search
//
// 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 loudAndRich(richer [][]int, quiet []int) []int {

}
```

Kotlin Solution:

```
class Solution {
fun loudAndRich(richer: Array<IntArray>, quiet: IntArray): IntArray {

}
}
```

Swift Solution:

```
class Solution {
func loudAndRich(_ richer: [[Int]], _ quiet: [Int]) -> [Int] {

}
}
```

Rust Solution:

```
// Problem: Loud and Rich
// Difficulty: Medium
// Tags: array, graph, sort, search
//
// 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 loud_and_rich(richer: Vec<Vec<i32>>, quiet: Vec<i32>) -> Vec<i32> {

}
}
```

Ruby Solution:

```
# @param {Integer[][]} richer
# @param {Integer[]} quiet
# @return {Integer[]}
def loud_and_rich(richer, quiet)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[][] $richer
* @param Integer[] $quiet
* @return Integer[]
*/
function loudAndRich($richer, $quiet) {

}
}
```

Dart Solution:

```
class Solution {
List<int> loudAndRich(List<List<int>> richer, List<int> quiet) {

}
}
```

Scala Solution:

```
object Solution {
def loudAndRich(richer: Array[Array[Int]], quiet: Array[Int]): Array[Int] = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec loud_and_rich(richer :: [[integer]], quiet :: [integer]) :: [integer]
def loud_and_rich(richer, quiet) do

end
end
```

Erlang Solution:

```
-spec loud_and_rich(Richer :: [[integer()]], Quiet :: [integer()]) ->
[integer()].
loud_and_rich(Richer, Quiet) ->
.
```

Racket Solution:

```
(define/contract (loud-and-rich richer quiet)
(-> (listof (listof exact-integer?)) (listof exact-integer?) (listof
exact-integer?))
)
```