Problem 2225: Find Players With Zero or One Losses

Problem Information

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

Problem Description

You are given an integer array

matches

where

matches[i] = [winner

i

, loser

i

]

indicates that the player

winner

i

defeated player

loser

i

in a match.

Return

a list

answer

of size

2

where:

answer[0]

is a list of all players that have

not

lost any matches.

answer[1]

is a list of all players that have lost exactly

one

match.

The values in the two lists should be returned in

increasing

order.

Note:

You should only consider the players that have played

at least one

match.

The testcases will be generated such that

no

two matches will have the

same

outcome.

Example 1:

Input:

matches = [[1,3],[2,3],[3,6],[5,6],[5,7],[4,5],[4,8],[4,9],[10,4],[10,9]]

Output:

[[1,2,10],[4,5,7,8]]

Explanation:

Players 1, 2, and 10 have not lost any matches. Players 4, 5, 7, and 8 each have lost one match. Players 3, 6, and 9 each have lost two matches. Thus, answer[0] = [1,2,10] and answer[1] = [4,5,7,8].

Example 2:

Input:

matches = [[2,3],[1,3],[5,4],[6,4]]

Output:

[[1,2,5,6],[]]

Explanation:

Players 1, 2, 5, and 6 have not lost any matches. Players 3 and 4 each have lost two matches. Thus, answer[0] = [1,2,5,6] and answer[1] = [].

Constraints:

1 <= matches.length <= 10

5

matches[i].length == 2

1 <= winner

i

, loser

i

<= 10

5

winner

i

!= loser

i

All

matches[i]

are

unique

.

Code Snippets

C++:

```
class Solution {
public:
vector<vector<int>> findWinners(vector<vector<int>>& matches) {

}
};
```

Java:

```
class Solution {
public List<List<Integer>> findWinners(int[][] matches) {

}
}
```

Python3:

```
class Solution:
def findWinners(self, matches: List[List[int]]) -> List[List[int]]:
```

Python:

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

JavaScript:

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

};
```

TypeScript:

```
function findWinners(matches: number[][]): number[][] {

};
```

C#:

```
public class Solution {
public IList<IList<int>> FindWinners(int[][] matches) {

}
}
```

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** findWinners(int** matches, int matchesSize, int* matchesColSize, int*
returnSize, int** returnColumnSizes) {

}
```

Go:

```
func findWinners(matches [][]int) [][]int {

}
```

Kotlin:

```
class Solution {
fun findWinners(matches: Array<IntArray>): List<List<Int>> {

}
}
```

Swift:

```
class Solution {
func findWinners(_ matches: [[Int]]) -> [[Int]] {

}
}
```

Rust:

```
impl Solution {
pub fn find_winners(matches: Vec<Vec<i32>>) -> Vec<Vec<i32>> {

}
}
```

Ruby:

```
# @param {Integer[][]} matches
# @return {Integer[][]}
def find_winners(matches)

end
```

PHP:

```
class Solution {

/**
* @param Integer[][] $matches
* @return Integer[][]
*/
function findWinners($matches) {

}
```

```
}
```

Dart:

```
class Solution {
List<List<int>> findWinners(List<List<int>> matches) {

}
}
```

Scala:

```
object Solution {
def findWinners(matches: Array[Array[Int]]): List[List[Int]] = {

}
}
```

Elixir:

```
defmodule Solution do
@spec find_winners(matches :: [[integer]]) :: [[integer]]
def find_winners(matches) do

end
end
```

Erlang:

```
-spec find_winners(Matches :: [[integer()]]) -> [[integer()]].
find_winners(Matches) ->
.
```

Racket:

```
(define/contract (find-winners matches)
(-> (listof (listof exact-integer?)) (listof (listof exact-integer?)))
)
```

Solutions

C++ Solution:

```
/*
* Problem: Find Players With Zero or One Losses
* Difficulty: Medium
* Tags: array, hash, sort
*
* 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>> findWinners(vector<vector<int>>& matches) {

}
};
```

Java Solution:

```
/**
* Problem: Find Players With Zero or One Losses
* Difficulty: Medium
* Tags: array, hash, sort
*
* 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>> findWinners(int[][] matches) {

}
}
```

Python3 Solution:

```
"""
Problem: Find Players With Zero or One Losses
Difficulty: Medium
Tags: array, hash, sort
```

```
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 findWinners(self, matches: List[List[int]]) -> List[List[int]]:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Find Players With Zero or One Losses
* Difficulty: Medium
* Tags: array, hash, sort
*
* 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[][]} matches
* @return {number[][]}
*/
var findWinners = function(matches) {

};
```

TypeScript Solution:

```
/**
* Problem: Find Players With Zero or One Losses
* Difficulty: Medium
* Tags: array, hash, sort
*
* 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 findWinners(matches: number[][]): number[][] {

};
```

C# Solution:

```
/*
* Problem: Find Players With Zero or One Losses
* Difficulty: Medium
* Tags: array, hash, sort
*
* 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>> FindWinners(int[][] matches) {

}
}
```

C Solution:

```
/*
* Problem: Find Players With Zero or One Losses
* Difficulty: Medium
* Tags: array, hash, sort
*
* 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** findWinners(int** matches, int matchesSize, int* matchesColSize, int*
returnSize, int** returnColumnSizes) {

}
```

Go Solution:

```
// Problem: Find Players With Zero or One Losses
// Difficulty: Medium
// Tags: array, hash, sort
//
// 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 findWinners(matches [][]int) [][]int {

}
```

Kotlin Solution:

```
class Solution {
fun findWinners(matches: Array<IntArray>): List<List<Int>> {

}
}
```

Swift Solution:

```
class Solution {
func findWinners(_ matches: [[Int]]) -> [[Int]] {

}
```

```
}
```

Rust Solution:

```
// Problem: Find Players With Zero or One Losses
// Difficulty: Medium
// Tags: array, hash, sort
//
// 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 find_winners(matches: Vec<Vec<i32>>) -> Vec<Vec<i32>> {

}
}
```

Ruby Solution:

```
# @param {Integer[][]} matches
# @return {Integer[][]}
def find_winners(matches)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[][] $matches
* @return Integer[][]
*/
function findWinners($matches) {

}
}
```

Dart Solution:

```
class Solution {
List<List<int>> findWinners(List<List<int>> matches) {

}
}
```

Scala Solution:

```
object Solution {
def findWinners(matches: Array[Array[Int]]): List[List[Int]] = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec find_winners(matches :: [[integer]]) :: [[integer]]
def find_winners(matches) do

end
end
```

Erlang Solution:

```
-spec find_winners(Matches :: [[integer()]]) -> [[integer()]].
find_winners(Matches) ->
.
```

Racket Solution:

```
(define/contract (find-winners matches)
(-> (listof (listof exact-integer?)) (listof (listof exact-integer?)))
)
```