Problem 1583: Count Unhappy Friends

Problem Information

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

Problem Description

You are given a list of

preferences

for

n

friends, where

n

is always

even

.

For each person

i

,

preferences[i]

contains a list of friends

sorted

in the

order of preference

. In other words, a friend earlier in the list is more preferred than a friend later in the list. Friends in each list are denoted by integers from

0

to

n-1

.

All the friends are divided into pairs. The pairings are given in a list

pairs

, where

pairs[i] = [x

i

, y

i

]

denotes

x

i

is paired with

y

i

and

y

i

is paired with

x

i

.

However, this pairing may cause some of the friends to be unhappy. A friend

x

is unhappy if

x

is paired with

y

and there exists a friend

u

who is paired with

v

but:

x

prefers

u

over

y

, and

u

prefers

x

over

v

.

Return

the number of unhappy friends

.

Example 1:

Input:

n = 4, preferences = [[1, 2, 3], [3, 2, 0], [3, 1, 0], [1, 2, 0]], pairs = [[0, 1], [2, 3]]

Output:

2

Explanation:

Friend 1 is unhappy because: - 1 is paired with 0 but prefers 3 over 0, and - 3 prefers 1 over 2.
Friend 3 is unhappy because: - 3 is paired with 2 but prefers 1 over 2, and - 1 prefers 3 over 0.
Friends 0 and 2 are happy.

Example 2:

Input:

n = 2, preferences = [[1], [0]], pairs = [[1, 0]]

Output:

0

Explanation:

Both friends 0 and 1 are happy.

Example 3:

Input:

n = 4, preferences = [[1, 3, 2], [2, 3, 0], [1, 3, 0], [0, 2, 1]], pairs = [[1, 3], [0, 2]]

Output:

4

Constraints:

2 <= n <= 500

n

is even.

preferences.length == n

preferences[i].length == n - 1

0 <= preferences[i][j] <= n - 1

preferences[i]

does not contain

i

.

All values in

preferences[i]

are unique.

pairs.length == n/2

pairs[i].length == 2

x

i

!= y

i

0 <= x

i

, y

i

<= n - 1

Each person is contained in

exactly one

pair.

Code Snippets

C++:

```
class Solution {
public:
int unhappyFriends(int n, vector<vector<int>>& preferences,
vector<vector<int>>& pairs) {

}
};
```

Java:

```
class Solution {
public int unhappyFriends(int n, int[][] preferences, int[][] pairs) {

}
}
```

Python3:

```
class Solution:
def unhappyFriends(self, n: int, preferences: List[List[int]], pairs:
List[List[int]]) -> int:
```

Python:

```
class Solution(object):
def unhappyFriends(self, n, preferences, pairs):
"""
:type n: int
:type preferences: List[List[int]]
:type pairs: List[List[int]]
:rtype: int
"""
```

JavaScript:

```
/**
* @param {number} n
* @param {number[][]} preferences
* @param {number[][]} pairs
* @return {number}
*/
var unhappyFriends = function(n, preferences, pairs) {

};
```

TypeScript:

```
function unhappyFriends(n: number, preferences: number[][], pairs:
number[][]): number {

};
```

C#:

```
public class Solution {
public int UnhappyFriends(int n, int[][] preferences, int[][] pairs) {

}
}
```

C:

```
int unhappyFriends(int n, int** preferences, int preferencesSize, int*
preferencesColSize, int** pairs, int pairsSize, int* pairsColSize) {
```

```
}
```

Go:

```
func unhappyFriends(n int, preferences [][]int, pairs [][]int) int {

}
```

Kotlin:

```
class Solution {
fun unhappyFriends(n: Int, preferences: Array<IntArray>, pairs:
Array<IntArray>): Int {

}
}
```

Swift:

```
class Solution {
func unhappyFriends(_ n: Int, _ preferences: [[Int]], _ pairs: [[Int]]) ->
Int {

}
}
```

Rust:

```
impl Solution {
pub fn unhappy_friends(n: i32, preferences: Vec<Vec<i32>>, pairs:
Vec<Vec<i32>>) -> i32 {

}
}
```

Ruby:

```
# @param {Integer} n
# @param {Integer[][]} preferences
# @param {Integer[][]} pairs
# @return {Integer}
def unhappy_friends(n, preferences, pairs)
```

```
end
```

PHP:

```
class Solution {

/**
* @param Integer $n
* @param Integer[][] $preferences
* @param Integer[][] $pairs
* @return Integer
*/
function unhappyFriends($n, $preferences, $pairs) {

}
}
```

Dart:

```
class Solution {
int unhappyFriends(int n, List<List<int>> preferences, List<List<int>> pairs)
{

}
}
```

Scala:

```
object Solution {
def unhappyFriends(n: Int, preferences: Array[Array[Int]], pairs:
Array[Array[Int]]): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec unhappy_friends(n :: integer, preferences :: [[integer]], pairs ::
[[integer]]) :: integer
def unhappy_friends(n, preferences, pairs) do
```

```
end
end
```

Erlang:

```
-spec unhappy_friends(N :: integer(), Preferences :: [[integer()]], Pairs ::
[[integer()]]) -> integer().
unhappy_friends(N, Preferences, Pairs) ->
.
```

Racket:

```
(define/contract (unhappy-friends n preferences pairs)
(-> exact-integer? (listof (listof exact-integer?)) (listof (listof
exact-integer?)) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Count Unhappy Friends
* Difficulty: Medium
* Tags: array, sort
*
* 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 unhappyFriends(int n, vector<vector<int>>& preferences,
vector<vector<int>>& pairs) {

}
};
```

Java Solution:

```
/**
* Problem: Count Unhappy Friends
* Difficulty: Medium
* Tags: array, sort
*
* 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 unhappyFriends(int n, int[][] preferences, int[][] pairs) {

}
}
```

Python3 Solution:

```
"""
Problem: Count Unhappy Friends
Difficulty: Medium
Tags: array, sort

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 unhappyFriends(self, n: int, preferences: List[List[int]], pairs:
List[List[int]]) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def unhappyFriends(self, n, preferences, pairs):
"""
:type n: int
:type preferences: List[List[int]]
```

```
:type pairs: List[List[int]]
:rtype: int
"""
```

JavaScript Solution:

```
/**
* Problem: Count Unhappy Friends
* Difficulty: Medium
* Tags: array, sort
*
* 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} n
* @param {number[][]} preferences
* @param {number[][]} pairs
* @return {number}
*/
var unhappyFriends = function(n, preferences, pairs) {

};
```

TypeScript Solution:

```
/**
* Problem: Count Unhappy Friends
* Difficulty: Medium
* Tags: array, sort
*
* 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 unhappyFriends(n: number, preferences: number[][], pairs:
number[][]): number {
```

```
};
```

C# Solution:

```
/*
* Problem: Count Unhappy Friends
* Difficulty: Medium
* Tags: array, sort
*
* 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 UnhappyFriends(int n, int[][] preferences, int[][] pairs) {

}
}
```

C Solution:

```
/*
* Problem: Count Unhappy Friends
* Difficulty: Medium
* Tags: array, sort
*
* 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
*/

int unhappyFriends(int n, int** preferences, int preferencesSize, int*
preferencesColSize, int** pairs, int pairsSize, int* pairsColSize) {

}
```

Go Solution:

```
// Problem: Count Unhappy Friends
// Difficulty: Medium
```

```
// Tags: array, sort
//
// 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 unhappyFriends(n int, preferences [][]int, pairs [][]int) int {

}
```

Kotlin Solution:

```
class Solution {
fun unhappyFriends(n: Int, preferences: Array<IntArray>, pairs:
Array<IntArray>): Int {

}
}
```

Swift Solution:

```
class Solution {
func unhappyFriends(_ n: Int, _ preferences: [[Int]], _ pairs: [[Int]]) ->
Int {

}
}
```

Rust Solution:

```
// Problem: Count Unhappy Friends
// Difficulty: Medium
// Tags: array, sort
//
// 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 unhappy_friends(n: i32, preferences: Vec<Vec<i32>>, pairs:
Vec<Vec<i32>>) -> i32 {
```

```
}
}
```

Ruby Solution:

```
# @param {Integer} n
# @param {Integer[][]} preferences
# @param {Integer[][]} pairs
# @return {Integer}
def unhappy_friends(n, preferences, pairs)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer $n
* @param Integer[][] $preferences
* @param Integer[][] $pairs
* @return Integer
*/
function unhappyFriends($n, $preferences, $pairs) {

}
}
```

Dart Solution:

```
class Solution {
int unhappyFriends(int n, List<List<int>> preferences, List<List<int>> pairs)
{

}
}
```

Scala Solution:

```
object Solution {
def unhappyFriends(n: Int, preferences: Array[Array[Int]], pairs:
Array[Array[Int]]): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec unhappy_friends(n :: integer, preferences :: [[integer]], pairs ::
[[integer]]) :: integer
def unhappy_friends(n, preferences, pairs) do

end
end
```

Erlang Solution:

```
-spec unhappy_friends(N :: integer(), Preferences :: [[integer()]], Pairs ::
[[integer()]]) -> integer().
unhappy_friends(N, Preferences, Pairs) ->
.
```

Racket Solution:

```
(define/contract (unhappy-friends n preferences pairs)
(-> exact-integer? (listof (listof exact-integer?)) (listof (listof
exact-integer?)) exact-integer?)
)
```