Problem 1947: Maximum Compatibility Score Sum

Problem Information

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

Problem Description

There is a survey that consists of

n

questions where each question's answer is either

0

(no) or

1

(yes).

The survey was given to

m

students numbered from

0

to

m - 1

and

m

mentors numbered from

0

to

m - 1

. The answers of the students are represented by a 2D integer array

students

where

students[i]

is an integer array that contains the answers of the

i

th

student (

0-indexed

). The answers of the mentors are represented by a 2D integer array

mentors

where

mentors[j]

is an integer array that contains the answers of the

j

th

mentor (

0-indexed

).

Each student will be assigned to

one

mentor, and each mentor will have

one

student assigned to them. The

compatibility score

of a student-mentor pair is the number of answers that are the same for both the student and the mentor.

For example, if the student's answers were

[1,

0

,

1

]

and the mentor's answers were

[0,

0

,

1

]

, then their compatibility score is 2 because only the second and the third answers are the same.

You are tasked with finding the optimal student-mentor pairings to

maximize

the

sum of the compatibility scores

.

Given

students

and

mentors

, return

the

maximum compatibility score sum

that can be achieved.

Example 1:

Input:

students = [[1,1,0],[1,0,1],[0,0,1]], mentors = [[1,0,0],[0,0,1],[1,1,0]]

Output:

8

Explanation:

We assign students to mentors in the following way: - student 0 to mentor 2 with a compatibility score of 3. - student 1 to mentor 0 with a compatibility score of 2. - student 2 to mentor 1 with a compatibility score of 3. The compatibility score sum is 3 + 2 + 3 = 8.

Example 2:

Input:

students = [[0,0],[0,0],[0,0]], mentors = [[1,1],[1,1],[1,1]]

Output:

0

Explanation:

The compatibility score of any student-mentor pair is 0.

Constraints:

m == students.length == mentors.length

n == students[i].length == mentors[j].length

1 <= m, n <= 8

students[i][k]

is either

0

or

1

.

mentors[j][k]

is either

0

or

1

.

Code Snippets

C++:

```
class Solution {
public:
int maxCompatibilitySum(vector<vector<int>>& students, vector<vector<int>>&
mentors) {

}
};
```

Java:

```
class Solution {
public int maxCompatibilitySum(int[][] students, int[][] mentors) {

}
}
```

Python3:

```
class Solution:
def maxCompatibilitySum(self, students: List[List[int]], mentors:
List[List[int]]) -> int:
```

Python:

```
class Solution(object):
def maxCompatibilitySum(self, students, mentors):
"""
:type students: List[List[int]]
:type mentors: List[List[int]]
:rtype: int
"""
```

JavaScript:

```
/**
* @param {number[][]} students
* @param {number[][]} mentors
* @return {number}
*/
var maxCompatibilitySum = function(students, mentors) {

};
```

TypeScript:

```
function maxCompatibilitySum(students: number[][], mentors: number[][]):
number {

};
```

C#:

```
public class Solution {
public int MaxCompatibilitySum(int[][] students, int[][] mentors) {

}
}
```

C:

```
int maxCompatibilitySum(int** students, int studentsSize, int*
studentsColSize, int** mentors, int mentorsSize, int* mentorsColSize) {

}
```

Go:

```
func maxCompatibilitySum(students [][]int, mentors [][]int) int {

}
```

Kotlin:

```
class Solution {
fun maxCompatibilitySum(students: Array<IntArray>, mentors: Array<IntArray>):
Int {

}
}
```

Swift:

```
class Solution {
func maxCompatibilitySum(_ students: [[Int]], _ mentors: [[Int]]) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn max_compatibility_sum(students: Vec<Vec<i32>>, mentors: Vec<Vec<i32>>)
-> i32 {

}

}
```

Ruby:

```
# @param {Integer[][]} students
# @param {Integer[][]} mentors
# @return {Integer}
def max_compatibility_sum(students, mentors)

end
```

PHP:

```
class Solution {

/**
* @param Integer[][] $students
* @param Integer[][] $mentors
* @return Integer
*/
function maxCompatibilitySum($students, $mentors) {

}
}
```

Dart:

```
class Solution {
int maxCompatibilitySum(List<List<int>> students, List<List<int>> mentors) {

}
}
```

Scala:

```
object Solution {
def maxCompatibilitySum(students: Array[Array[Int]], mentors:
Array[Array[Int]]): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec max_compatibility_sum(students :: [[integer]], mentors :: [[integer]])
:: integer
def max_compatibility_sum(students, mentors) do

end
end
```

Erlang:

```
-spec max_compatibility_sum(Students :: [[integer()]], Mentors ::
[[integer()]]) -> integer().
max_compatibility_sum(Students, Mentors) ->
.
```

Racket:

```
(define/contract (max-compatibility-sum students mentors)
(-> (listof (listof exact-integer?)) (listof (listof exact-integer?))
exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Maximum Compatibility Score Sum
* Difficulty: Medium
* Tags: array, dp
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

class Solution {
public:
int maxCompatibilitySum(vector<vector<int>>& students, vector<vector<int>>&
mentors) {
}
};
```

Java Solution:

```
/**
* Problem: Maximum Compatibility Score Sum
* Difficulty: Medium
* Tags: array, dp
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

class Solution {
public int maxCompatibilitySum(int[][] students, int[][] mentors) {

}
}
```

Python3 Solution:

```
"""
Problem: Maximum Compatibility Score Sum
Difficulty: Medium
Tags: array, dp

Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(n) or O(n * m) for DP table
"""

class Solution:
def maxCompatibilitySum(self, students: List[List[int]], mentors:
List[List[int]]) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def maxCompatibilitySum(self, students, mentors):
"""
:type students: List[List[int]]
:type mentors: List[List[int]]
:rtype: int
"""
```

JavaScript Solution:

```
/**
* Problem: Maximum Compatibility Score Sum
* Difficulty: Medium
* Tags: array, dp
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

/**
* @param {number[][]} students
* @param {number[][]} mentors
* @return {number}
*/
var maxCompatibilitySum = function(students, mentors) {

};
```

TypeScript Solution:

```
/**
* Problem: Maximum Compatibility Score Sum
* Difficulty: Medium
* Tags: array, dp
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

function maxCompatibilitySum(students: number[][], mentors: number[][]):
```

```
number {

};
```

C# Solution:

```
/*
* Problem: Maximum Compatibility Score Sum
* Difficulty: Medium
* Tags: array, dp
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

public class Solution {
public int MaxCompatibilitySum(int[][] students, int[][] mentors) {

}
}
```

C Solution:

```
/*
* Problem: Maximum Compatibility Score Sum
* Difficulty: Medium
* Tags: array, dp
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

int maxCompatibilitySum(int** students, int studentsSize, int*
studentsColSize, int** mentors, int mentorsSize, int* mentorsColSize) {

}
```

Go Solution:

```
// Problem: Maximum Compatibility Score Sum
// Difficulty: Medium
// Tags: array, dp
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) or O(n * m) for DP table

func maxCompatibilitySum(students [][]int, mentors [][]int) int {

}
```

Kotlin Solution:

```
class Solution {
fun maxCompatibilitySum(students: Array<IntArray>, mentors: Array<IntArray>): Int {

}
}
```

Swift Solution:

```
class Solution {
func maxCompatibilitySum(_ students: [[Int]], _ mentors: [[Int]]) -> Int {

}
}
```

Rust Solution:

```
// Problem: Maximum Compatibility Score Sum
// Difficulty: Medium
// Tags: array, dp
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) or O(n * m) for DP table

impl Solution {
pub fn max_compatibility_sum(students: Vec<Vec<i32>>, mentors: Vec<Vec<i32>>) -> i32 {


}
}
```

Ruby Solution:

```
# @param {Integer[][]} students
# @param {Integer[][]} mentors
# @return {Integer}
def max_compatibility_sum(students, mentors)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[][] $students
* @param Integer[][] $mentors
* @return Integer
*/
function maxCompatibilitySum($students, $mentors) {

}
}
```

Dart Solution:

```
class Solution {
int maxCompatibilitySum(List<List<int>> students, List<List<int>> mentors) {

}
}
```

Scala Solution:

```
object Solution {
def maxCompatibilitySum(students: Array[Array[Int]], mentors:
Array[Array[Int]]): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec max_compatibility_sum(students :: [[integer]], mentors :: [[integer]])
:: integer
def max_compatibility_sum(students, mentors) do

end
end
```

Erlang Solution:

```
-spec max_compatibility_sum(Students :: [[integer()]], Mentors ::
[[integer()]]) -> integer().
max_compatibility_sum(Students, Mentors) ->
.
```

Racket Solution:

```
(define/contract (max-compatibility-sum students mentors)
(-> (listof (listof exact-integer?)) (listof (listof exact-integer?))
exact-integer?)
)
```