Problem 2352: Equal Row and Column Pairs

Problem Information

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

Problem Description

Given a

0-indexed

n x n

integer matrix

grid

,

return the number of pairs

(r

i

, c

j

)

such that row

r

i

and column

c

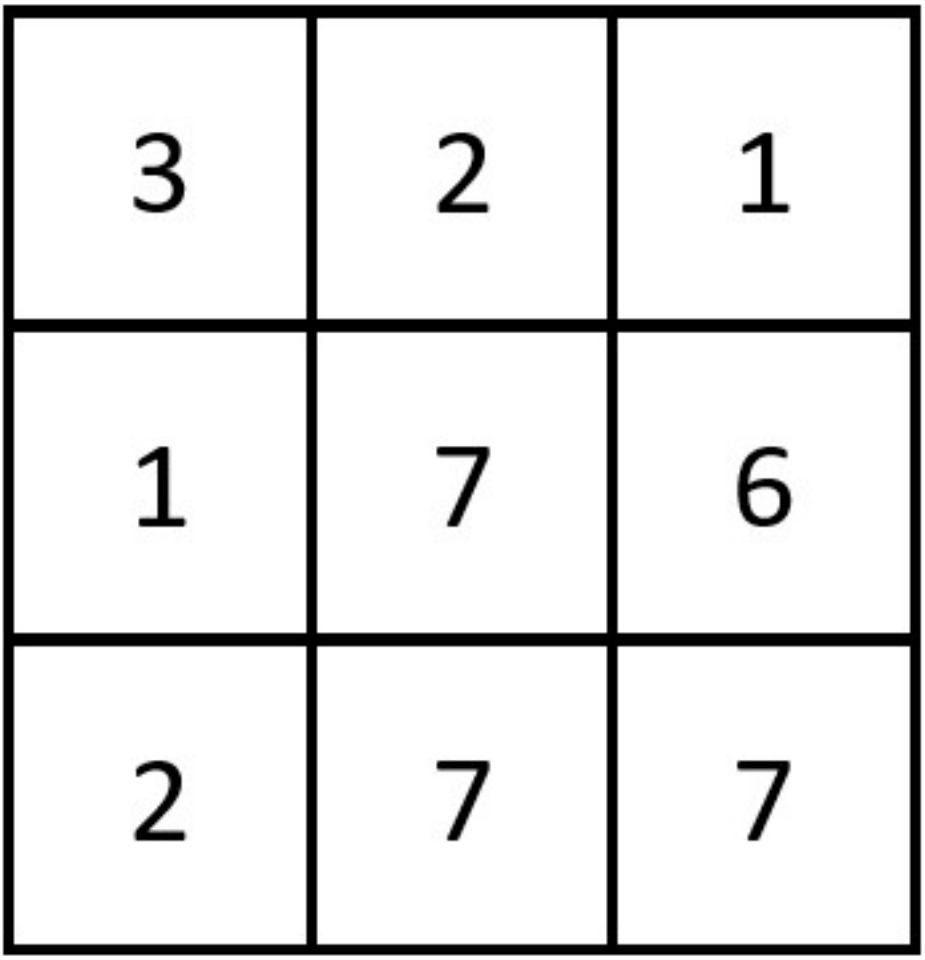
j

are equal

.

A row and column pair is considered equal if they contain the same elements in the same order (i.e., an equal array).

Example 1:



Input:

grid = [[3,2,1],[1,7,6],[2,7,7]]

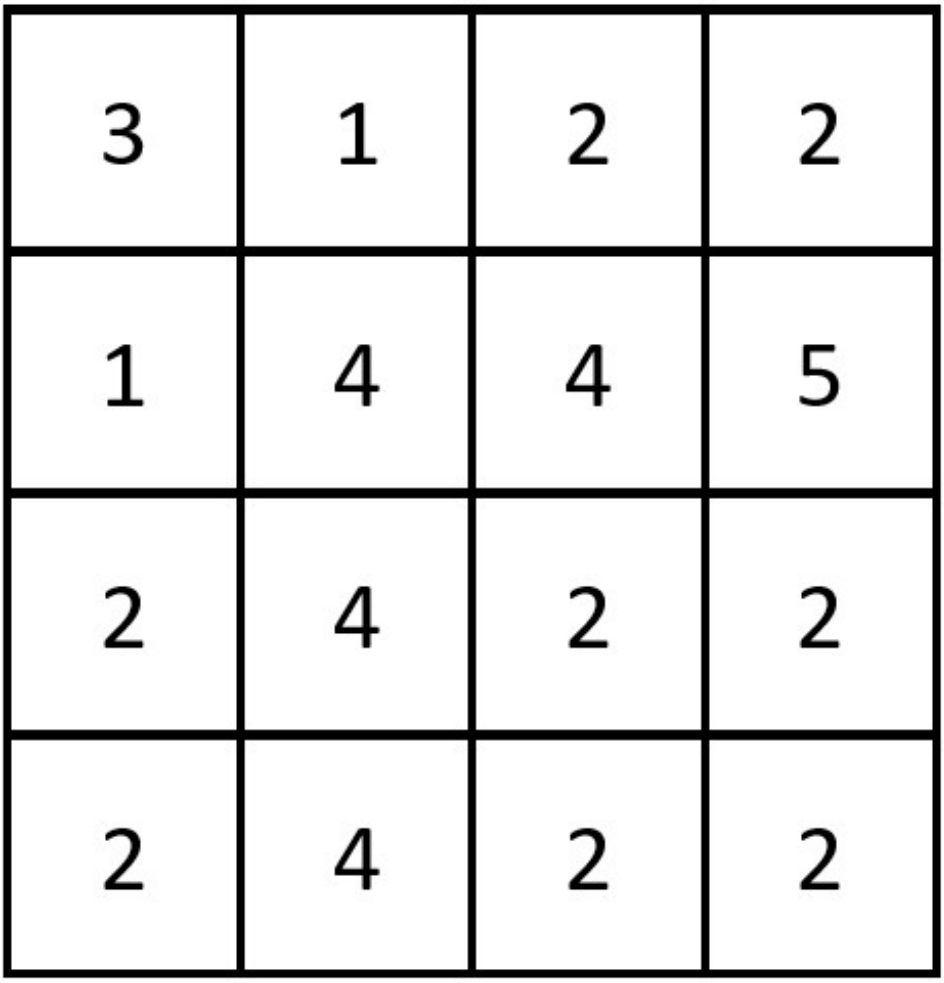
Output:

1

Explanation:

There is 1 equal row and column pair: - (Row 2, Column 1): [2,7,7]

Example 2:



Input:

grid = [[3,1,2,2],[1,4,4,5],[2,4,2,2],[2,4,2,2]]

Output:

3

Explanation:

There are 3 equal row and column pairs: - (Row 0, Column 0): [3,1,2,2] - (Row 2, Column 2): [2,4,2,2] - (Row 3, Column 2): [2,4,2,2]

Constraints:

n == grid.length == grid[i].length

1 <= n <= 200

1 <= grid[i][j] <= 10

5

Code Snippets

C++:

```
class Solution {
public:
int equalPairs(vector<vector<int>>& grid) {

}
};
```

Java:

```
class Solution {
public int equalPairs(int[][] grid) {

}
}
```

Python3:

```
class Solution:
def equalPairs(self, grid: List[List[int]]) -> int:
```

Python:

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

JavaScript:

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

};
```

TypeScript:

```
function equalPairs(grid: number[][]): number {

};
```

C#:

```
public class Solution {
public int EqualPairs(int[][] grid) {

}
}
```

C:

```
int equalPairs(int** grid, int gridSize, int* gridColSize) {

}
```

Go:

```
func equalPairs(grid [][]int) int {

}
```

Kotlin:

```
class Solution {
fun equalPairs(grid: Array<IntArray>): Int {

}
}
```

Swift:

```
class Solution {
func equalPairs(_ grid: [[Int]]) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn equal_pairs(grid: Vec<Vec<i32>>) -> i32 {

}
}
```

Ruby:

```
# @param {Integer[][]} grid
# @return {Integer}
def equal_pairs(grid)

end
```

PHP:

```
class Solution {

/**
* @param Integer[][] $grid
* @return Integer
*/
function equalPairs($grid) {

}
}
```

Dart:

```
class Solution {
int equalPairs(List<List<int>> grid) {

}
}
```

Scala:

```
object Solution {
def equalPairs(grid: Array[Array[Int]]): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec equal_pairs(grid :: [[integer]]) :: integer
def equal_pairs(grid) do

end
end
```

Erlang:

```
-spec equal_pairs(Grid :: [[integer()]]) -> integer().
equal_pairs(Grid) ->
.
```

Racket:

```
(define/contract (equal-pairs grid)
(-> (listof (listof exact-integer?)) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Equal Row and Column Pairs
* Difficulty: Medium
* Tags: array, 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:
int equalPairs(vector<vector<int>>& grid) {

}
};
```

Java Solution:

```
/**
* Problem: Equal Row and Column Pairs
* Difficulty: Medium
* Tags: array, 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 int equalPairs(int[][] grid) {

}
}
```

Python3 Solution:

```
"""
Problem: Equal Row and Column Pairs
Difficulty: Medium
Tags: array, 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 equalPairs(self, grid: List[List[int]]) -> int:
# TODO: Implement optimized solution
```

```
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Equal Row and Column Pairs
* Difficulty: Medium
* Tags: array, 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[][]} grid
* @return {number}
*/
var equalPairs = function(grid) {

};
```

TypeScript Solution:

```
/**
* Problem: Equal Row and Column Pairs
* Difficulty: Medium
* Tags: array, 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 equalPairs(grid: number[][]): number {

};
```

C# Solution:

```
/*
* Problem: Equal Row and Column Pairs
* Difficulty: Medium
* Tags: array, 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 int EqualPairs(int[][] grid) {

}
}
```

C Solution:

```
/*
* Problem: Equal Row and Column Pairs
* Difficulty: Medium
* Tags: array, 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
*/

int equalPairs(int** grid, int gridSize, int* gridColSize) {

}
```

Go Solution:

```
// Problem: Equal Row and Column Pairs
// Difficulty: Medium
// Tags: array, 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 equalPairs(grid [][]int) int {

}
```

Kotlin Solution:

```
class Solution {
fun equalPairs(grid: Array<IntArray>): Int {

}
}
```

Swift Solution:

```
class Solution {
func equalPairs(_ grid: [[Int]]) -> Int {

}
}
```

Rust Solution:

```
// Problem: Equal Row and Column Pairs
// Difficulty: Medium
// Tags: array, 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 equal_pairs(grid: Vec<Vec<i32>>) -> i32 {

}
```

```
}
```

Ruby Solution:

```
# @param {Integer[][]} grid
# @return {Integer}
def equal_pairs(grid)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[][] $grid
* @return Integer
*/
function equalPairs($grid) {

}
}
```

Dart Solution:

```
class Solution {
int equalPairs(List<List<int>> grid) {

}
}
```

Scala Solution:

```
object Solution {
def equalPairs(grid: Array[Array[Int]]): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec equal_pairs(grid :: [[integer]]) :: integer
def equal_pairs(grid) do

end
end
```

Erlang Solution:

```
-spec equal_pairs(Grid :: [[integer()]]) -> integer().
equal_pairs(Grid) ->
.
```

Racket Solution:

```
(define/contract (equal-pairs grid)
(-> (listof (listof exact-integer?)) exact-integer?)
)
```