Problem 3725: Count Ways to Choose Coprime Integers from Rows

Problem Information

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

Problem Description

You are given a

m x n

matrix

mat

of positive integers.

Return an integer denoting the number of ways to choose

exactly one

integer from each row of

mat

such that the

greatest common divisor

of all chosen integers is 1.

Since the answer may be very large, return it

modulo

10

9

+ 7

.

Example 1:

Input:

mat = [[1,2],[3,4]]

Output:

3

Explanation:

Chosen integer in the first row

Chosen integer in the second row

Greatest common divisor of chosen integers

1

3

1

1

4

1

2

3

1

2

4

2

3 of these combinations have a greatest common divisor of 1. Therefore, the answer is 3.

Example 2:

Input:

mat = [[2,2],[2,2]]

Output:

0

Explanation:

Every combination has a greatest common divisor of 2. Therefore, the answer is 0.

Constraints:

1 <= m == mat.length <= 150

1 <= n == mat[i].length <= 150

1 <= mat[i][j] <= 150

Code Snippets

C++:

```
class Solution {
public:
int countCoprime(vector<vector<int>>& mat) {

}
};
```

Java:

```
class Solution {
public int countCoprime(int[][] mat) {

}
}
```

Python3:

```
class Solution:
def countCoprime(self, mat: List[List[int]]) -> int:
```

Python:

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

JavaScript:

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

};
```

TypeScript:

```
function countCoprime(mat: number[][]): number {

};
```

C#:

```
public class Solution {
public int CountCoprime(int[][] mat) {

}
}
```

C:

```
int countCoprime(int** mat, int matSize, int* matColSize) {

}
```

Go:

```
func countCoprime(mat [][]int) int {

}
```

Kotlin:

```
class Solution {
fun countCoprime(mat: Array<IntArray>): Int {

}
}
```

Swift:

```
class Solution {
func countCoprime(_ mat: [[Int]]) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn count_coprime(mat: Vec<Vec<i32>>) -> i32 {

}
}
```

Ruby:

```
# @param {Integer[][]} mat
# @return {Integer}
def count_coprime(mat)

end
```

PHP:

```
class Solution {

/**
* @param Integer[][] $mat
* @return Integer
*/
function countCoprime($mat) {

}
}
```

Dart:

```
class Solution {
int countCoprime(List<List<int>> mat) {

}
}
```

Scala:

```
object Solution {
def countCoprime(mat: Array[Array[Int]]): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec count_coprime(mat :: [[integer]]) :: integer
def count_coprime(mat) do

end
end
```

Erlang:

```
-spec count_coprime(Mat :: [[integer()]]) -> integer().
count_coprime(Mat) ->
.
```

Racket:

```
(define/contract (count-coprime mat)
(-> (listof (listof exact-integer?)) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Count Ways to Choose Coprime Integers from Rows
* Difficulty: Hard
* Tags: array, dp, math
*
* 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 countCoprime(vector<vector<int>>& mat) {

}
};
```

Java Solution:

```
/**
* Problem: Count Ways to Choose Coprime Integers from Rows
* Difficulty: Hard
* Tags: array, dp, math
*
* 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 countCoprime(int[][] mat) {

}
}
```

Python3 Solution:

```
"""
Problem: Count Ways to Choose Coprime Integers from Rows
Difficulty: Hard
Tags: array, dp, math

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

Python Solution:

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

```
"""
```

JavaScript Solution:

```
/**
* Problem: Count Ways to Choose Coprime Integers from Rows
* Difficulty: Hard
* Tags: array, dp, math
*
* 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[][]} mat
* @return {number}
*/
var countCoprime = function(mat) {

};
```

TypeScript Solution:

```
/**
* Problem: Count Ways to Choose Coprime Integers from Rows
* Difficulty: Hard
* Tags: array, dp, math
*
* 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 countCoprime(mat: number[][]): number {

};
```

C# Solution:

```
/*
* Problem: Count Ways to Choose Coprime Integers from Rows
* Difficulty: Hard
* Tags: array, dp, math
*
* 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 CountCoprime(int[][] mat) {

}
}
```

C Solution:

```
/*
* Problem: Count Ways to Choose Coprime Integers from Rows
* Difficulty: Hard
* Tags: array, dp, math
*
* 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 countCoprime(int** mat, int matSize, int* matColSize) {

}
```

Go Solution:

```
// Problem: Count Ways to Choose Coprime Integers from Rows
// Difficulty: Hard
// Tags: array, dp, math
//
// 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 countCoprime(mat [][]int) int {

}
```

Kotlin Solution:

```
class Solution {
fun countCoprime(mat: Array<IntArray>): Int {

}
}
```

Swift Solution:

```
class Solution {
func countCoprime(_ mat: [[Int]]) -> Int {

}
}
```

Rust Solution:

```
// Problem: Count Ways to Choose Coprime Integers from Rows
// Difficulty: Hard
// Tags: array, dp, math
//
// 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 count_coprime(mat: Vec<Vec<i32>>) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer[][]} mat
# @return {Integer}
def count_coprime(mat)
```

```
end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[][] $mat
* @return Integer
*/
function countCoprime($mat) {

}
}
```

Dart Solution:

```
class Solution {
int countCoprime(List<List<int>> mat) {

}
}
```

Scala Solution:

```
object Solution {
def countCoprime(mat: Array[Array[Int]]): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec count_coprime(mat :: [[integer]]) :: integer
def count_coprime(mat) do

end
end
```

Erlang Solution:

```
-spec count_coprime(Mat :: [[integer()]]) -> integer().
count_coprime(Mat) ->
.
```

Racket Solution:

```
(define/contract (count-coprime mat)
(-> (listof (listof exact-integer?)) exact-integer?)
)
```