Problem 840: Magic Squares In Grid

Problem Information

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

Problem Description

A

3 x 3

magic square

is a

3 x 3

grid filled with distinct numbers

from

1

to

9 such that each row, column, and both diagonals all have the same sum.

Given a

row x col

grid

of integers, how many

3 x 3

magic square subgrids are there?

Note: while a magic square can only contain numbers from 1 to 9,

grid

may contain numbers up to 15.

Example 1:

4	3	8	4
9	5	1	9
2	7	6	2

Input:

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

Output:

1

Explanation:

The following subgrid is a 3 x 3 magic square:

4	3	8
9	5	1
2	7	6

while this one is not:

3	8	4
5	1	9
7	6	2

In total, there is only one magic square inside the given grid.

Example 2:

Input:

grid = [[8]]

Output:

0

Constraints:

row == grid.length

col == grid[i].length

1 <= row, col <= 10

0 <= grid[i][j] <= 15

Code Snippets

C++:

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

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

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

}
}
```

C:

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

}
```

Go:

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

```
}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

/**
* @param Integer[][] $grid
* @return Integer
*/
```

```
function numMagicSquaresInside($grid) {

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

```
(define/contract (num-magic-squares-inside grid)
(-> (listof (listof exact-integer?)) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Magic Squares In Grid
* Difficulty: Medium
* Tags: array, math, 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 numMagicSquaresInside(vector<vector<int>>& grid) {

}
};
```

Java Solution:

```
/**
* Problem: Magic Squares In Grid
* Difficulty: Medium
* Tags: array, math, 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 numMagicSquaresInside(int[][] grid) {

}
}
```

Python3 Solution:

```
"""
Problem: Magic Squares In Grid
Difficulty: Medium
Tags: array, math, 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 numMagicSquaresInside(self, grid: List[List[int]]) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Magic Squares In Grid
* Difficulty: Medium
* Tags: array, math, 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 numMagicSquaresInside = function(grid) {
```

```
};
```

TypeScript Solution:

```
/**
* Problem: Magic Squares In Grid
* Difficulty: Medium
* Tags: array, math, 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 numMagicSquaresInside(grid: number[][]): number {

};
```

C# Solution:

```
/*
* Problem: Magic Squares In Grid
* Difficulty: Medium
* Tags: array, math, 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 NumMagicSquaresInside(int[][] grid) {

}
}
```

C Solution:

```
/*
* Problem: Magic Squares In Grid
* Difficulty: Medium
```

```
* Tags: array, math, 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 numMagicSquaresInside(int** grid, int gridSize, int* gridColSize) {

}
```

Go Solution:

```
// Problem: Magic Squares In Grid
// Difficulty: Medium
// Tags: array, math, 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 numMagicSquaresInside(grid [][]int) int {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Magic Squares In Grid
// Difficulty: Medium
// Tags: array, math, 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 num_magic_squares_inside(grid: Vec<Vec<i32>>) -> i32 {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

```
}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

```
(define/contract (num-magic-squares-inside grid)
(-> (listof (listof exact-integer?)) exact-integer?)
)
```