840. Magic Squares In Grid

Solved

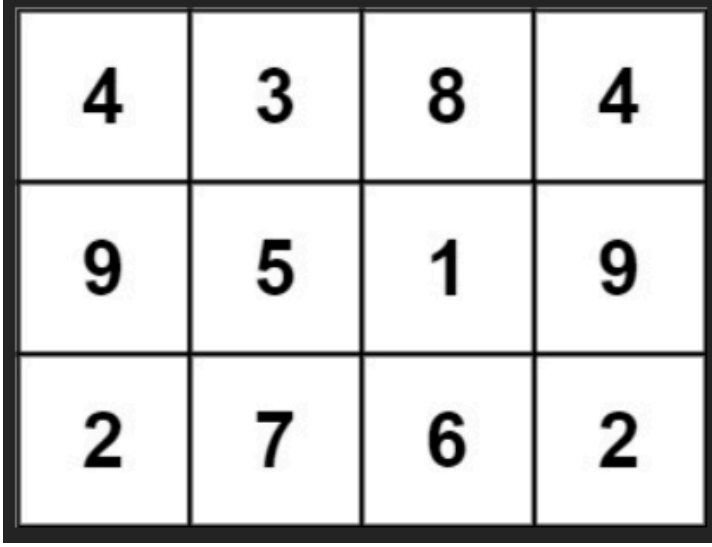
Medium | Topics | Companies

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:



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`

Python:

```
class Solution:
    def numMagicSquaresInside(self, grid: List[List[int]]) -> int:
        count = 0
        rows, cols = len(grid), len(grid[0])

        for i in range(rows - 2):
            for j in range(cols - 2):
                if self.is_valid_magic_square(grid, i, j):
                    count += 1

        return count

    def is_valid_magic_square(self, grid: List[List[int]], start_row: int, start_col: int) -> bool:
        # Create a boolean array to track the presence of each number (1-9)
        num_presence = [False] * 10

        # Check each element in the 3x3 subgrid
        for i in range(start_row, start_row + 3):
            for j in range(start_col, start_col + 3):
                num = grid[i][j]
                if num < 1 or num > 9 or num_presence[num]:
```

```
                return False
            num_presence[num] = True

        # Check if the sums of rows, columns, and diagonals are equal
        target_sum = grid[start_row][start_col] + grid[start_row][start_col + 1] +
grid[start_row][start_col + 2]
        for i in range(3):
            if self.get_row_sum(grid, start_row + i, start_col) != target_sum or \
               self.get_col_sum(grid, start_row, start_col + i) != target_sum:
                return False

        diagonal_sum1 = grid[start_row][start_col] + grid[start_row + 1][start_col + 1] +
grid[start_row + 2][start_col + 2]
        diagonal_sum2 = grid[start_row + 2][start_col] + grid[start_row + 1][start_col + 1] +
grid[start_row][start_col + 2]
        return diagonal_sum1 == target_sum and diagonal_sum2 == target_sum

    def get_row_sum(self, grid: List[List[int]], row: int, start_col: int) -> int:
        return grid[row][start_col] + grid[row][start_col + 1] + grid[row][start_col + 2]

    def get_col_sum(self, grid: List[List[int]], start_row: int, col: int) -> int:
        return grid[start_row][col] + grid[start_row + 1][col] + grid[start_row + 2][col]
```

JavaScript:

```
/**
 * @param {number[][]} grid
 * @return {number}
 */
var numMagicSquaresInside = function(grid) {
    let count = 0;
    const rows = grid.length;
    const cols = grid[0].length;

    for (let i = 0; i <= rows - 3; i++) {
        for (let j = 0; j <= cols - 3; j++) {
            if (isValidMagicSquare(grid, i, j)) {
                count++;
            }
        }
    }

    return count;
};
```

```
/**
 * @param {number[][]} grid
 * @param {number} startRow
 * @param {number} startCol
 * @return {boolean}
 */
function isValidMagicSquare(grid, startRow, startCol) {
    // Create a boolean array to track the presence of each number (1-9)
    const numPresence = new Array(10).fill(false);

    // Check each element in the 3x3 subgrid
    for (let i = startRow; i < startRow + 3; i++) {
        for (let j = startCol; j < startCol + 3; j++) {
            const num = grid[i][j];
            if (num < 1 || num > 9 || numPresence[num]) {
                return false;
            }
            numPresence[num] = true;
        }
    }

    // Check if the sums of rows, columns, and diagonals are equal
    const targetSum = grid[startRow][startCol] + grid[startRow][startCol + 1] +
grid[startRow][startCol + 2];
    for (let i = 0; i < 3; i++) {
        if (getRowSum(grid, startRow + i, startCol) !== targetSum ||
            getColSum(grid, startRow, startCol + i) !== targetSum) {
            return false;
        }
    }

    const diagonalSum1 = grid[startRow][startCol] + grid[startRow + 1][startCol + 1] +
grid[startRow + 2][startCol + 2];
    const diagonalSum2 = grid[startRow + 2][startCol] + grid[startRow + 1][startCol + 1] +
grid[startRow][startCol + 2];
    return diagonalSum1 === targetSum && diagonalSum2 === targetSum;
}

/**
 * @param {number[][]} grid
 * @param {number} row
 * @param {number} startCol
 * @return {number}
 */
```

```
function getRowSum(grid, row, startCol) {
    return grid[row][startCol] + grid[row][startCol + 1] + grid[row][startCol + 2];
}

/**
 * @param {number[][]} grid
 * @param {number} startRow
 * @param {number} col
 * @return {number}
 */
function getColSum(grid, startRow, col) {
    return grid[startRow][col] + grid[startRow + 1][col] + grid[startRow + 2][col];
}
```

Java:

```
class Solution {
    public int numMagicSquaresInside(int[][] grid) {
        int rows = grid.length;
        int cols = grid[0].length;
        if (rows < 3 || cols < 3) {
            return 0;
        }
        int count = 0;
        for (int i = 0; i <= rows - 3; i++) {
            for (int j = 0; j <= cols - 3; j++) {
                if (isValidMagicSquare(grid, i, j)) {
                    count++;
                }
            }
        }
        return count;
    }

    private boolean isValidMagicSquare(int[][] grid, int startRow, int startCol) {
        boolean[] numPresence = new boolean[10];
        int targetSum = 0;

        // Check for distinct numbers and calculate sums
        for (int i = 0; i < 3; i++) {
            int rowSum = 0;
            int colSum = 0;
            for (int j = 0; j < 3; j++) {
                int num = grid[startRow + i][startCol + j];
                if (num < 1 || num > 9 || numPresence[num]) {
```

```
                return false;
            }
            numPresence[num] = true;
            rowSum += num;
            colSum += grid[startRow + j][startCol + i];
        }
        if (i == 0) {
            targetSum = rowSum;
        } else if (rowSum != targetSum || colSum != targetSum) {
            return false;
        }
    }

    // Check diagonals
    int mainDiagonalSum = grid[startRow][startCol] + grid[startRow + 1][startCol + 1] +
grid[startRow + 2][startCol + 2];
    int antiDiagonalSum = grid[startRow][startCol + 2] + grid[startRow + 1][startCol + 1] +
grid[startRow + 2][startCol];

    return mainDiagonalSum == targetSum && antiDiagonalSum == targetSum;
  }
}
```