840. Magic Squares In Grid

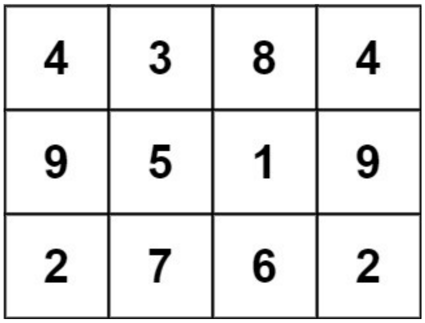
A `3 x 3` **magic square** is a `3 x 3` grid filled with distinct numbers **from** 1 **to** 9 such that each row, column, and both diagonals all have the same sum.

Given a `row x col` `grid` of integers, how many `3 x 3` contiguous 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.

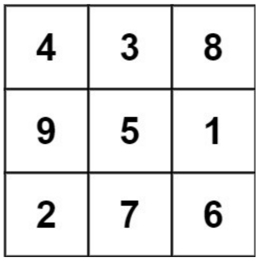
Example 1:

```
Input: grid = [[4,3,8,4],[9,5,1,9],
[2,7,6,2]]
Output: 1
```



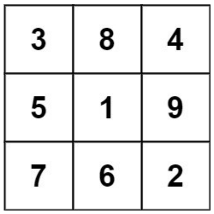
4	3	8	4
9	5	1	9
2	7	6	2

```
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:

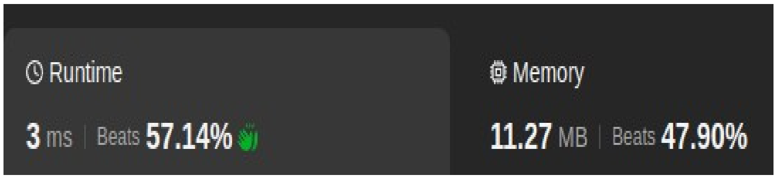
```
Input: grid = [[8]]
Output: 0
```

Constraints:

- `row == grid.length`
- `col == grid[i].length`
- `1 <= row, col <= 10`
- `0 <= grid[i][j] <= 15`

840. Magic Squares In Grid

```
/*
   Brute force
Time complexity:
O(nm*(9+9+9+3+9+9))=O(48nm)=O(nm)
   Space complexity: O(9+9)=O(1)
```



Runtime

3 ms | Beats 57.14%

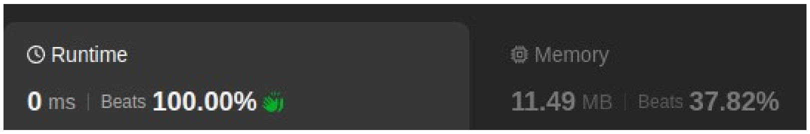
Memory

11.27 MB | Beats 47.90%

```
*/
class Solution {
   public:
      bool is_magic_square(std::vector<std::vector<int>>& square){
         // Check if digits from 1 to 9 are distinct
         bool exists[10]={false};
         for(int i=0;i<3;++i){
            for(int j=0;j<3;++j){
               int x=square[i][j];
               if(x>9 || exists[x]) return false;
               exists[x]=true;
            }
         }

         // Check if all digits from 1 to 9 exist
         for(int i=1;i<=9;++i) if(!exists[i]) return false;

         // Check sums
         // Compute first sum
         int row_sum=0;
         for(int i=0;i<3;++i) row_sum+=square[0][i];

         // Sum of each row
         for(int i=0;i<3;++i){
            int s=0;
            for(int j=0;j<3;++j){
               s+=square[i][j];
            }
            if(s!=row_sum) return false;
         }
```

```
          // Sum of each column
            for(int i=0;i<3;++i){
                int s=0;
                for(int j=0;j<3;++j){
                    s+=square[j][i];
                }
                if(s!=row_sum) return false;
            }

            // Sum of the diagonal
            int sd1=square[0][0]+square[1][1]+square[2][2];

            // Sum of the anti-diagonal
            int sd2=square[0][2]+square[1][1]+square[2][0];

            if(sd1!=row_sum || sd2!=row_sum) return false;

            // If all checks are good
            return true;
        }

        int numMagicSquaresInside(std::vector<std::vector<int>>& grid) {
            int n=grid.size();
            int m=grid[0].size();
            int ans=0;
            for(int i=0;i<=n-3;++i){
                for(int j=0;j<=m-3;++j){
                    if(j+2>m-1) break;
                    if(i+2>n-1) return ans;
                    std::vector<std::vector<int>> square(3,std::vector<int>(3));
                    for(int k=0;k<3;++k){
                        for(int l=0;l<3;++l){
                            square[k][l]=grid[i+k][j+l];
                        }
                    }

                    if (is_magic_square(square)) ans++;
                }
            }
            return ans;
        }
};
```

840. Magic Squares In Grid



Runtime

0 ms | Beats 100.00%

Memory

11.49 MB | Beats 37.82%

```
/*
    Use 3x3 magic square properties
    Time complexity:
O(nm*(9+9))=O(18nm)=O(nm)
    Space complexity: O(9+3+3)=O(1)
*/

class Solution {
    public:
        bool is_magic_square(std::vector<std::vector<int>>& square){
            // If 5 is not in middle, it can not be a magic square
            if(square[1][1] != 5) return false;

            // Computes sums, and check if digits from 1 to 9
            // are all present once.
            int row_sum[3]={0},col_sum[3]={0};
            int mask=511;
            for(int i=0;i<3;++i){
                for(int j=0;j<3;++j){
                    int x=square[i][j];
                    if(x<=0 || x>9) return false;
                    row_sum[i]+=x;
                    col_sum[j]+=x;
                    mask=(mask & (~(1<<(x-1))));
                }
            }
            // mask !=0, means it exists a digit that exists than one, or
            // it exists a digit that does not a exist.
            if(mask!=0) return false;

            // sums of all rows, and all columns must be 15
            if (!all_of(row_sum, row_sum+3,[](int sum){return sum==15;})) return false;
            if (!all_of(col_sum, col_sum+3,[](int sum){return sum==15;})) return false;

            // It all checks above ar good
            return true;
        }
int numMagicSquaresInside(std::vector<std::vector<int>>& grid) {
            int n=grid.size();
            int m=grid[0].size();

            int ans=0;
            for(int i=0;i<=n-3;++i){
                for(int j=0;j<=m-3;++j){
                    if(j+2>m-1) break;
                    if(i+2>n-1) return ans;
                    std::vector<std::vector<int>> square(3,std::vector<int>(3));
                    for(int k=0;k<3;++k){
                        for(int l=0;l<3;++l){
                            square[k][l]=grid[i+k][j+l];
                        }
                    }
                    if (is_magic_square(square)) ans++;
                }
            }
            return ans;
        }
};
```

840. Magic Squares In Grid

Runtime
4 ms | Beats 38.66%

Memory
13.62 MB | Beats 7.14%

```
/*
    Use 3x3 magic square properties+Math observation
    Time complexity: O(nm*(9+15+15))=O(39nm)=O(nm)
    Space complexity: O(9+15+15)=O(1)
*/

class Solution {
    public:
        bool is_magic_square(std::vector<std::vector<int>>& square){
            if(square[1][1] != 5) return false;

            std::string spiral1="438167294381672";
            std::string spiral2="927618349276183";

            std::string s=std::to_string(square[0][0])+std::to_string(square[0][1])+std::to_string(square[0][2])
               +std::to_string(square[1][2])+std::to_string(square[2][2])
               +std::to_string(square[2][1])+std::to_string(square[2][0])
               +std::to_string(square[1][0]);

            if(spiral1.find(s)==std::string::npos && spiral2.find(s)==std::string::npos) return false;

            return true;
        }
int numMagicSquaresInside(std::vector<std::vector<int>>& grid) {
            int n=grid.size();
            int m=grid[0].size();

            int ans=0;
            for(int i=0;i<=n-3;++i){
                for(int j=0;j<=m-3;++j){
                    if(j+2>m-1) break;
                    if(i+2>n-1) return ans;
                    std::vector<std::vector<int>> square(3,std::vector<int>(3));
                    for(int k=0;k<3;++k){
                        for(int l=0;l<3;++l){
                            square[k][l]=grid[i+k][j+l];
                        }
                    }
                    if (is_magic_square(square)) ans++;
                }
            }
            return ans;
        }
};
```