Problem 1582: Special Positions in a Binary Matrix

Problem Information

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

Problem Description

Given an

m x n

binary matrix

mat

, return

the number of special positions in

mat

.

A position

(i, j)

is called

special

if

mat[i][j] == 1

and all other elements in row

i

and column

j

are

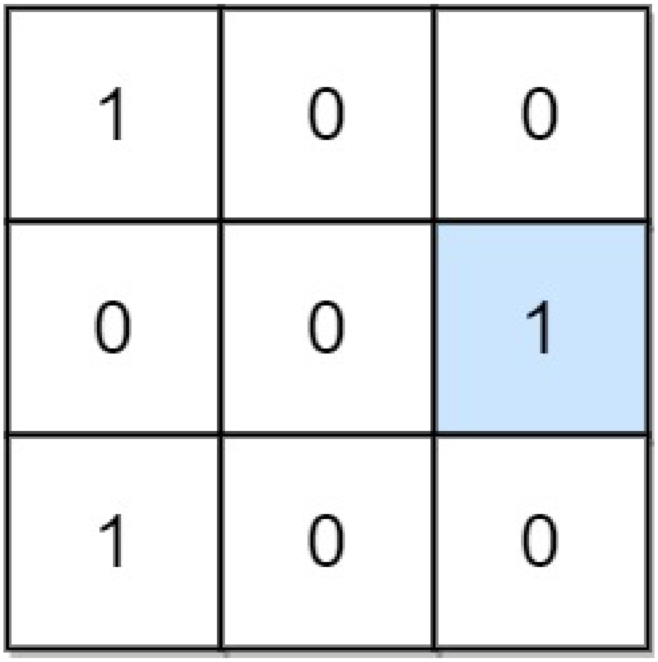
0

(rows and columns are

0-indexed

).

Example 1:



1	0	0
0	0	1
1	0	0

Input:

mat = [[1,0,0],[0,0,1],[1,0,0]]

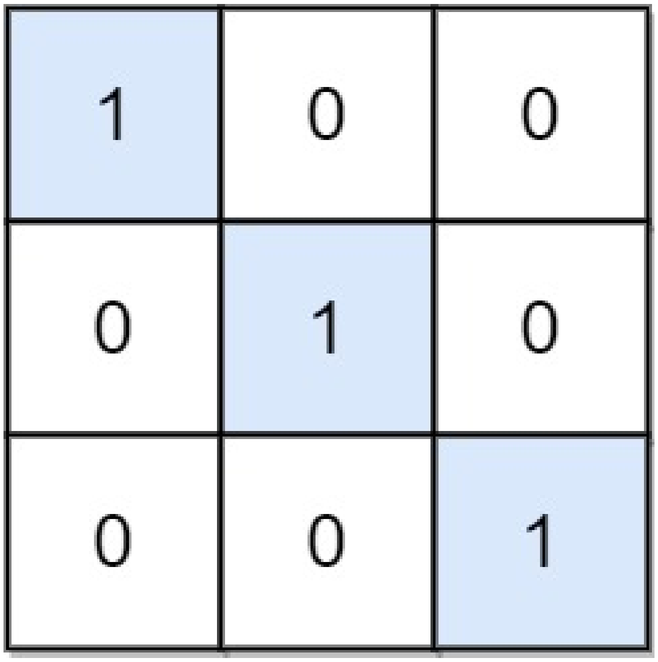
Output:

1

Explanation:

(1, 2) is a special position because mat[1][2] == 1 and all other elements in row 1 and column 2 are 0.

Example 2:



1	0	0
0	1	0
0	0	1

Input:

mat = [[1,0,0],[0,1,0],[0,0,1]]

Output:

3

Explanation:

(0, 0), (1, 1) and (2, 2) are special positions.

Constraints:

m == mat.length

n == mat[i].length

1 <= m, n <= 100

mat[i][j]

is either

0

or

1

.

Code Snippets

C++:

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

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

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

}
}
```

C:

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

}
```

Go:

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

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

```
(define/contract (num-special mat)
(-> (listof (listof exact-integer?)) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Special Positions in a Binary Matrix
* Difficulty: Easy
* Tags: array
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

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

}
};
```

Java Solution:

```
/**
* Problem: Special Positions in a Binary Matrix
* Difficulty: Easy
* Tags: array
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

class Solution {
public int numSpecial(int[][] mat) {

}
}
```

Python3 Solution:

```
"""
Problem: Special Positions in a Binary Matrix
Difficulty: Easy
Tags: array

Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(1) to O(n) depending on approach
"""

class Solution:
def numSpecial(self, mat: List[List[int]]) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Special Positions in a Binary Matrix
* Difficulty: Easy
* Tags: array
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

/**
```

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

};
```

TypeScript Solution:

```
/**
* Problem: Special Positions in a Binary Matrix
* Difficulty: Easy
* Tags: array
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

function numSpecial(mat: number[][]): number {

};
```

C# Solution:

```
/*
* Problem: Special Positions in a Binary Matrix
* Difficulty: Easy
* Tags: array
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

public class Solution {
public int NumSpecial(int[][] mat) {

}
}
```

C Solution:

```c
/*
* Problem: Special Positions in a Binary Matrix
* Difficulty: Easy
* Tags: array
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

int numSpecial(int** mat, int matSize, int* matColSize) {

}
```

Go Solution:

```go
// Problem: Special Positions in a Binary Matrix
// Difficulty: Easy
// Tags: array
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(1) to O(n) depending on approach

func numSpecial(mat [][]int) int {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

```swift
class Solution {
func numSpecial(_ mat: [[Int]]) -> Int {
}
}
```

Rust Solution:

```
// Problem: Special Positions in a Binary Matrix
// Difficulty: Easy
// Tags: array
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(1) to O(n) depending on approach

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

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

```
(define/contract (num-special mat)
(-> (listof (listof exact-integer?)) exact-integer?)
)
```