Problem 1267: Count Servers that Communicate

Problem Information

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

Problem Description

You are given a map of a server center, represented as a

m * n

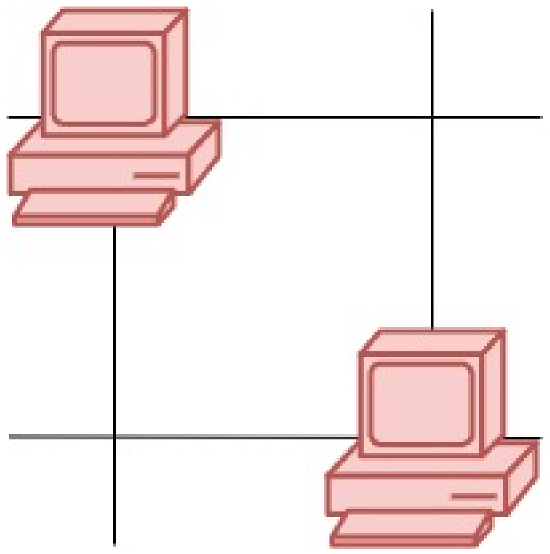
integer matrix

grid

, where 1 means that on that cell there is a server and 0 means that it is no server. Two servers are said to communicate if they are on the same row or on the same column.

Return the number of servers that communicate with any other server.

Example 1:



Input:

grid = [[1,0],[0,1]]

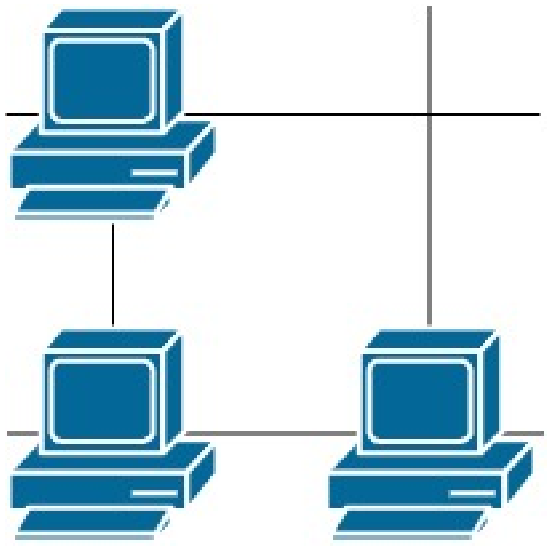
Output:

0

Explanation:

No servers can communicate with others.

Example 2:



Input:

grid = [[1,0],[1,1]]

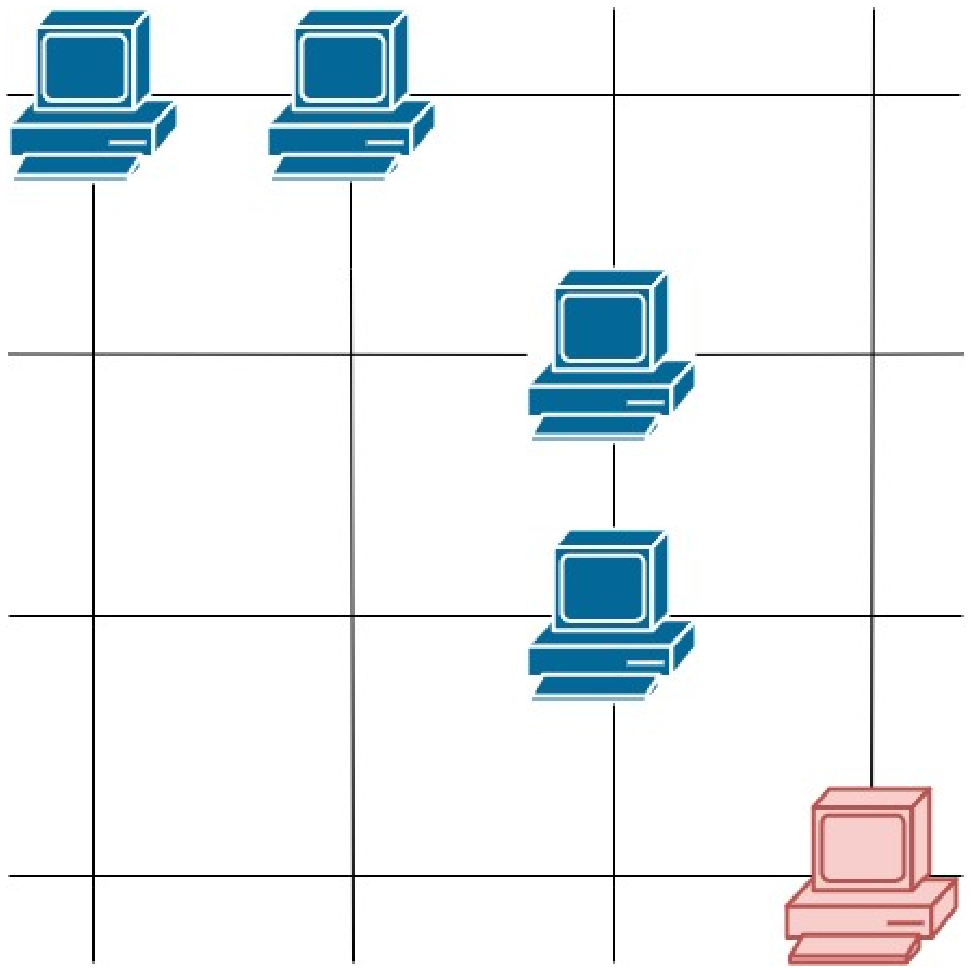
Output:

3

Explanation:

All three servers can communicate with at least one other server.

Example 3:



Input:

grid = [[1,1,0,0],[0,0,1,0],[0,0,1,0],[0,0,0,1]] Output:

4

Explanation:

The two servers in the first row can communicate with each other. The two servers in the third column can communicate with each other. The server at right bottom corner can't communicate with any other server.

Constraints:

m == grid.length

n == grid[i].length

1 <= m <= 250

1 <= n <= 250

grid[i][j] == 0 or 1

Code Snippets

C++:

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

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

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

}
}
```

C:

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

}
```

Go:

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

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

```
(define/contract (count-servers grid)
(-> (listof (listof exact-integer?)) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Count Servers that Communicate
* Difficulty: Medium
* Tags: array, graph, search
*
* 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 countServers(vector<vector<int>>& grid) {

}
};
```

Java Solution:

```
/**
* Problem: Count Servers that Communicate
* Difficulty: Medium
* Tags: array, graph, search
*
* 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 countServers(int[][] grid) {

}
}
```

Python3 Solution:

```
"""
Problem: Count Servers that Communicate
Difficulty: Medium
Tags: array, graph, search

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

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Count Servers that Communicate
* Difficulty: Medium
* Tags: array, graph, search
*
* 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[][]} grid
* @return {number}
*/
var countServers = function(grid) {

};
```

TypeScript Solution:

```
/**
* Problem: Count Servers that Communicate
* Difficulty: Medium
* Tags: array, graph, search
*
* 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 countServers(grid: number[][]): number {

};
```

C# Solution:

```
/*
* Problem: Count Servers that Communicate
* Difficulty: Medium
* Tags: array, graph, search
*
* 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 CountServers(int[][] grid) {

}
}
```

C Solution:

```
/*
* Problem: Count Servers that Communicate
* Difficulty: Medium
* Tags: array, graph, search
*
* 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 countServers(int** grid, int gridSize, int* gridColSize) {

}
```

Go Solution:

```
// Problem: Count Servers that Communicate
// Difficulty: Medium
// Tags: array, graph, search
//
// 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 countServers(grid [][]int) int {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

Rust Solution:

```
// Problem: Count Servers that Communicate
// Difficulty: Medium
// Tags: array, graph, search
//
// 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 count_servers(grid: Vec<Vec<i32>>) -> i32 {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

```
(define/contract (count-servers grid)
(-> (listof (listof exact-integer?)) exact-integer?)
)
```