Problem 2174: Remove All Ones With Row and Column Flips II

Problem Information

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

Problem Description

You are given a

0-indexed

m x n

binary

matrix

grid

.

In one operation, you can choose any

i

and

j

that meet the following conditions:

0 <= i < m

0 <= j < n

grid[i][j] == 1

and change the values of

all

cells in row

i

and column

j

to zero.

Return

the

minimum

number of operations needed to remove all

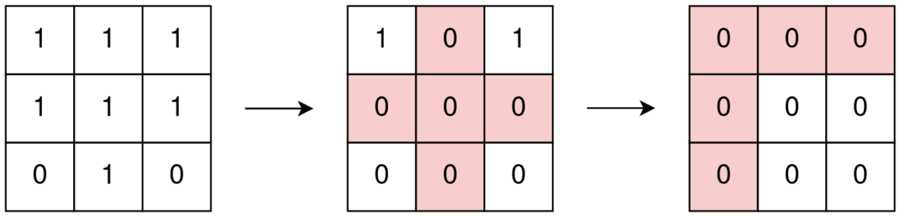
1

's from

grid

.

Example 1:



Input:

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

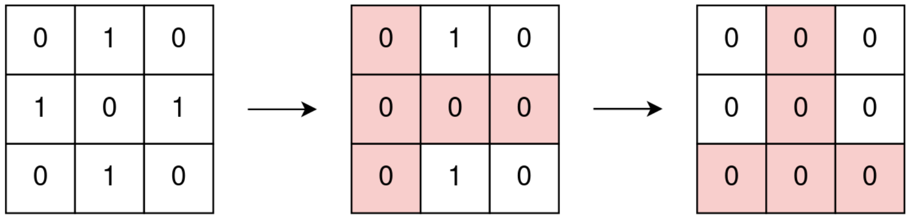
Output:

2

Explanation:

In the first operation, change all cell values of row 1 and column 1 to zero. In the second operation, change all cell values of row 0 and column 0 to zero.

Example 2:



Input:

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

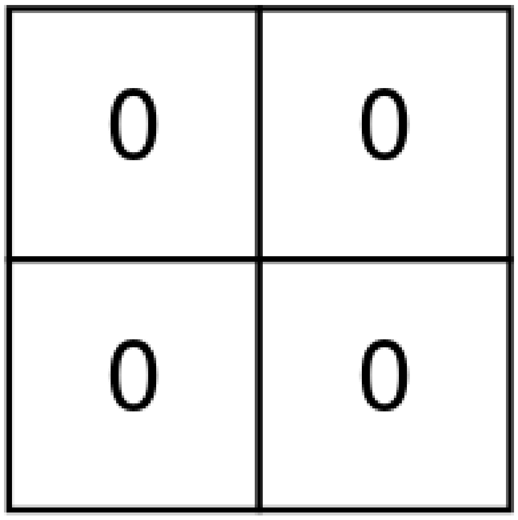
Output:

2

Explanation:

In the first operation, change all cell values of row 1 and column 0 to zero. In the second operation, change all cell values of row 2 and column 1 to zero. Note that we cannot perform an operation using row 1 and column 1 because grid[1][1] != 1.

Example 3:



Input:

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

Output:

0

Explanation:

There are no 1's to remove so return 0.

Constraints:

m == grid.length

n == grid[i].length

1 <= m, n <= 15

1 <= m * n <= 15

grid[i][j]

is either

0

or

1

.

Code Snippets

C++:

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

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

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

}
}
```

C:

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

}
```

Go:

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

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

```
(define/contract (remove-ones grid)
(-> (listof (listof exact-integer?)) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Remove All Ones With Row and Column Flips II
* Difficulty: Medium
* Tags: array, 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 removeOnes(vector<vector<int>>& grid) {

}
};
```

Java Solution:

```
/**
* Problem: Remove All Ones With Row and Column Flips II
* Difficulty: Medium
* Tags: array, 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 removeOnes(int[][] grid) {

}
}
```

Python3 Solution:

```
"""
Problem: Remove All Ones With Row and Column Flips II
Difficulty: Medium
Tags: array, 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 removeOnes(self, grid: List[List[int]]) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Remove All Ones With Row and Column Flips II
* Difficulty: Medium
* Tags: array, 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 removeOnes = function(grid) {
};
```

TypeScript Solution:

```
/**
* Problem: Remove All Ones With Row and Column Flips II
* Difficulty: Medium
* Tags: array, 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 removeOnes(grid: number[][]): number {

};
```

C# Solution:

```
/*
* Problem: Remove All Ones With Row and Column Flips II
* Difficulty: Medium
* Tags: array, 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 RemoveOnes(int[][] grid) {

}
}
```

C Solution:

```
/*
* Problem: Remove All Ones With Row and Column Flips II
* Difficulty: Medium
* Tags: array, 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 removeOnes(int** grid, int gridSize, int* gridColSize) {

}
```

Go Solution:

```
// Problem: Remove All Ones With Row and Column Flips II
// Difficulty: Medium
// Tags: array, 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 removeOnes(grid [][]int) int {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Remove All Ones With Row and Column Flips II
// Difficulty: Medium
// Tags: array, 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 remove_ones(grid: Vec<Vec<i32>>) -> i32 {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

```
(define/contract (remove-ones grid)
(-> (listof (listof exact-integer?)) exact-integer?)
)
```