Problem 2123: Minimum Operations to Remove Adjacent Ones in Matrix

Problem Information

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

Problem Description

You are given a

0-indexed

binary matrix

grid

. In one operation, you can flip any

1

in

grid

to be

0

.

A binary matrix is

well-isolated

if there is no

1

in the matrix that is

4-directionally connected

(i.e., horizontal and vertical) to another

1

.

Return

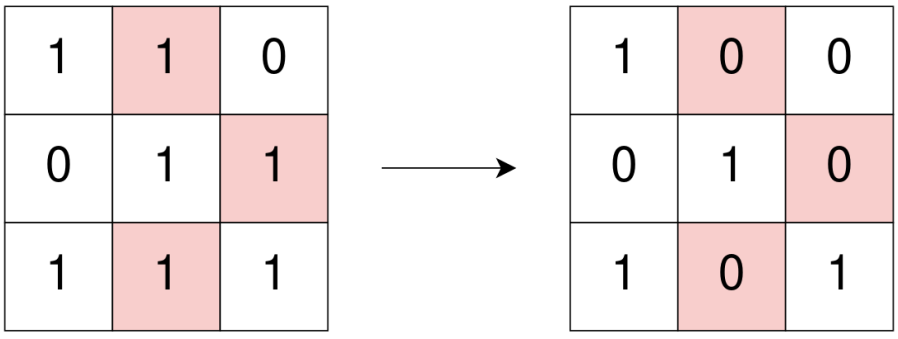
the minimum number of operations to make

grid

well-isolated

.

Example 1:



Input:

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

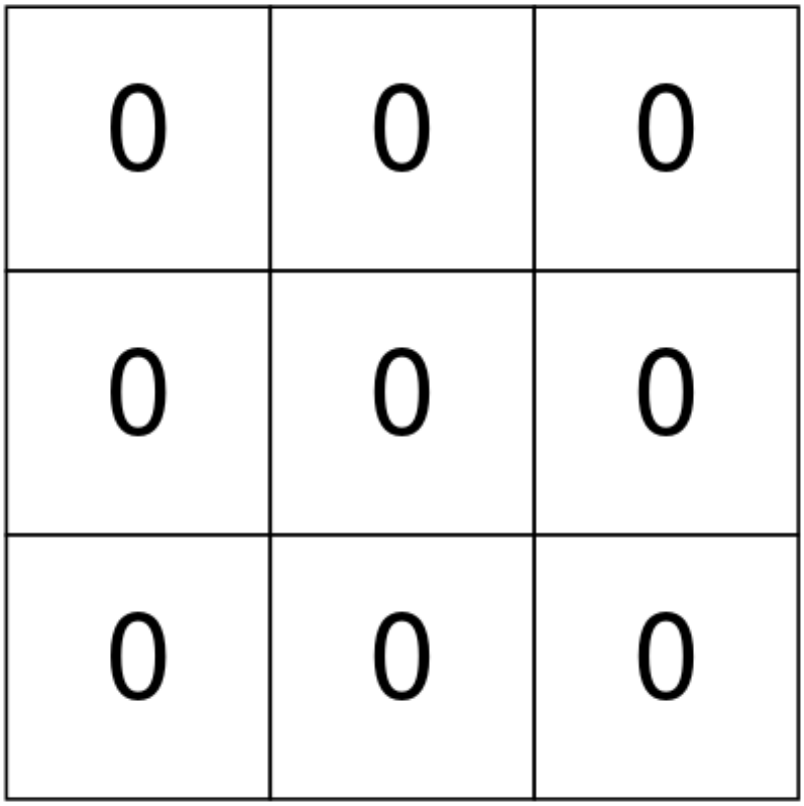
Output:

3

Explanation:

Use 3 operations to change grid[0][1], grid[1][2], and grid[2][1] to 0. After, no more 1's are 4-directionally connected and grid is well-isolated.

Example 2:



Input:

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

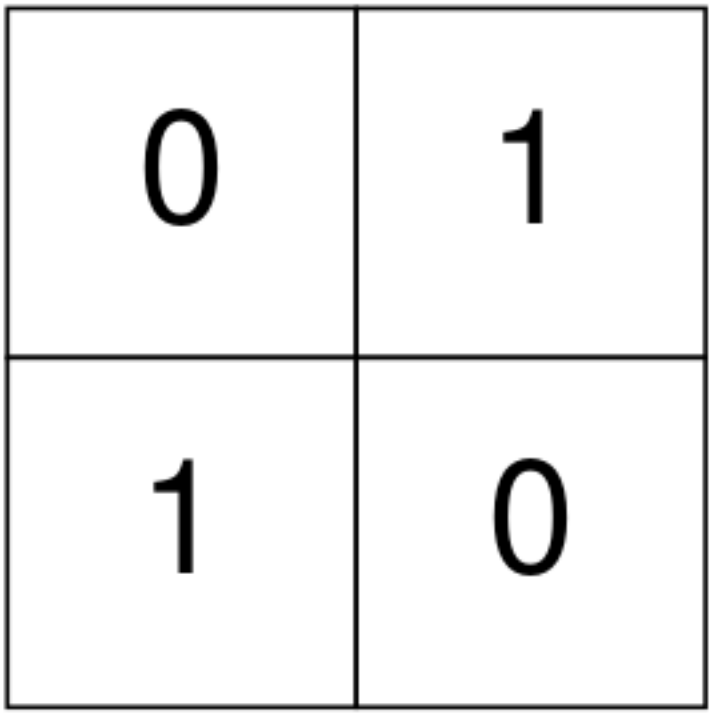
Output:

0

Explanation:

There are no 1's in grid and it is well-isolated. No operations were done so return 0.

Example 3:



Input:

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

Output:

0

Explanation:

None of the 1's are 4-directionally connected and grid is well-isolated. No operations were done so return 0.

Constraints:

m == grid.length

n == grid[i].length

1 <= m, n <= 300

grid[i][j]

is either

0

or

1

.

Code Snippets

C++:

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

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

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

}
}
```

C:

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

}
```

Go:

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

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

/**
* @param Integer[][] $grid
* @return Integer
```

```
*/
function minimumOperations($grid) {

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

```
(define/contract (minimum-operations grid)
(-> (listof (listof exact-integer?)) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Minimum Operations to Remove Adjacent Ones in Matrix
* Difficulty: Hard
* Tags: array, graph
*
* 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 minimumOperations(vector<vector<int>>& grid) {

}
};
```

Java Solution:

```
/**
* Problem: Minimum Operations to Remove Adjacent Ones in Matrix
* Difficulty: Hard
* Tags: array, graph
*
* 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 minimumOperations(int[][] grid) {

}
}
```

Python3 Solution:

```
"""
Problem: Minimum Operations to Remove Adjacent Ones in Matrix
Difficulty: Hard
Tags: array, graph

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

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Minimum Operations to Remove Adjacent Ones in Matrix
* Difficulty: Hard
* Tags: array, graph
*
* 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 minimumOperations = function(grid) {
```

```
};
```

TypeScript Solution:

```
/**
* Problem: Minimum Operations to Remove Adjacent Ones in Matrix
* Difficulty: Hard
* Tags: array, graph
*
* 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 minimumOperations(grid: number[][]): number {

};
```

C# Solution:

```
/*
* Problem: Minimum Operations to Remove Adjacent Ones in Matrix
* Difficulty: Hard
* Tags: array, graph
*
* 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 MinimumOperations(int[][] grid) {

}
}
```

C Solution:

```
/*
* Problem: Minimum Operations to Remove Adjacent Ones in Matrix
* Difficulty: Hard
```

```
* Tags: array, graph
*
* 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 minimumOperations(int** grid, int gridSize, int* gridColSize) {

}
```

Go Solution:

```
// Problem: Minimum Operations to Remove Adjacent Ones in Matrix
// Difficulty: Hard
// Tags: array, graph
//
// 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 minimumOperations(grid [][]int) int {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Minimum Operations to Remove Adjacent Ones in Matrix
// Difficulty: Hard
// Tags: array, graph
//
// 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 minimum_operations(grid: Vec<Vec<i32>>) -> i32 {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

```
}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

```
(define/contract (minimum-operations grid)
(-> (listof (listof exact-integer?)) exact-integer?)
)
```