Problem 2290: Minimum Obstacle Removal to Reach Corner

Problem Information

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

Problem Description

You are given a

0-indexed

2D integer array

grid

of size

m x n

. Each cell has one of two values:

0

represents an

empty

cell,

1

represents an

obstacle

that may be removed.

You can move up, down, left, or right from and to an empty cell.

Return

the

minimum

number of

obstacles

to

remove

so you can move from the upper left corner

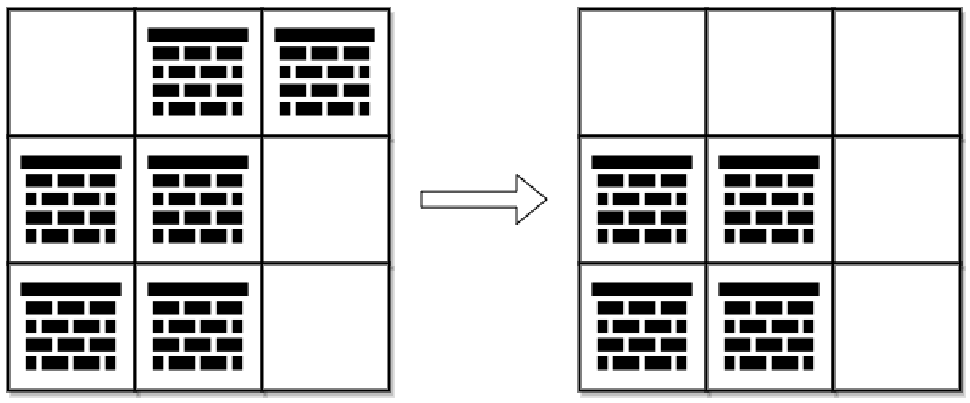
(0, 0)

to the lower right corner

(m - 1, n - 1)

.

Example 1:



Input:

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

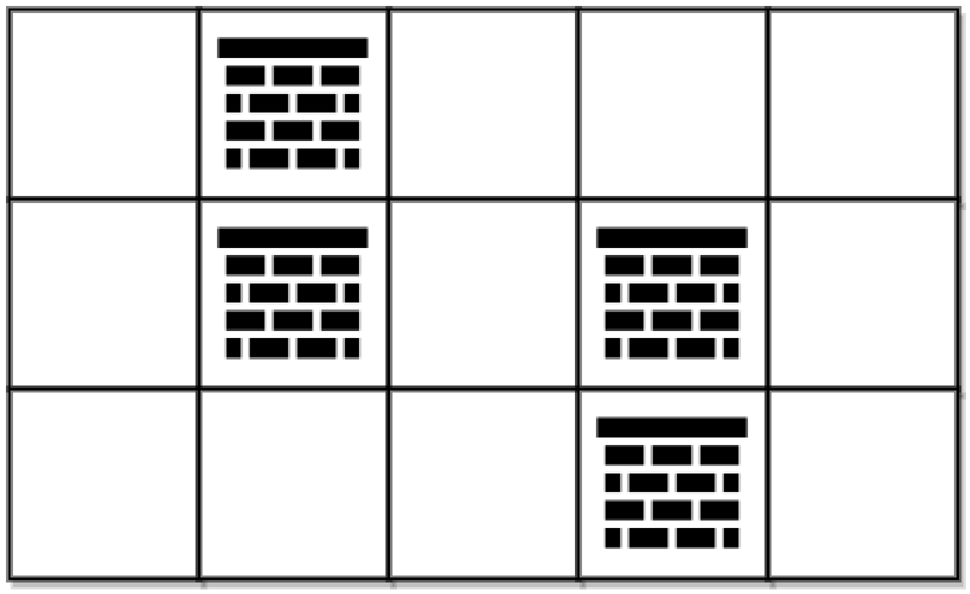
Output:

2

Explanation:

We can remove the obstacles at (0, 1) and (0, 2) to create a path from (0, 0) to (2, 2). It can be shown that we need to remove at least 2 obstacles, so we return 2. Note that there may be other ways to remove 2 obstacles to create a path.

Example 2:



Input:

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

Output:

0

Explanation:

We can move from (0, 0) to (2, 4) without removing any obstacles, so we return 0.

Constraints:

m == grid.length

n == grid[i].length

1 <= m, n <= 10

5

2 <= m * n <= 10

5

grid[i][j]

is either

0

or

1

.

grid[0][0] == grid[m - 1][n - 1] == 0

Code Snippets

C++:

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

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

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

}
}
```

C:

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

}
```

Go:

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

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

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

Solutions

C++ Solution:

```
/*
* Problem: Minimum Obstacle Removal to Reach Corner
* Difficulty: Hard
* Tags: array, tree, graph, search, queue, heap
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

class Solution {
public:
int minimumObstacles(vector<vector<int>>& grid) {
}
};
```

Java Solution:

```
/**
* Problem: Minimum Obstacle Removal to Reach Corner
* Difficulty: Hard
* Tags: array, tree, graph, search, queue, heap
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

class Solution {
public int minimumObstacles(int[][] grid) {

}
}
```

Python3 Solution:

```
"""
Problem: Minimum Obstacle Removal to Reach Corner
Difficulty: Hard
Tags: array, tree, graph, search, queue, heap

Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(h) for recursion stack where h is height
"""

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

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Minimum Obstacle Removal to Reach Corner
* Difficulty: Hard
* Tags: array, tree, graph, search, queue, heap
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

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

};
```

TypeScript Solution:

```
/**
* Problem: Minimum Obstacle Removal to Reach Corner
* Difficulty: Hard
* Tags: array, tree, graph, search, queue, heap
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

function minimumObstacles(grid: number[][]): number {

};
```

C# Solution:

```
/*
* Problem: Minimum Obstacle Removal to Reach Corner
* Difficulty: Hard
* Tags: array, tree, graph, search, queue, heap
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

public class Solution {
public int MinimumObstacles(int[][] grid) {

}
}
```

C Solution:

```
/*
* Problem: Minimum Obstacle Removal to Reach Corner
* Difficulty: Hard
* Tags: array, tree, graph, search, queue, heap
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

int minimumObstacles(int** grid, int gridSize, int* gridColSize) {

}
```

Go Solution:

```
// Problem: Minimum Obstacle Removal to Reach Corner
// Difficulty: Hard
// Tags: array, tree, graph, search, queue, heap
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(h) for recursion stack where h is height

func minimumObstacles(grid [][]int) int {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Minimum Obstacle Removal to Reach Corner
// Difficulty: Hard
// Tags: array, tree, graph, search, queue, heap
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(h) for recursion stack where h is height

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

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
```

```
end
```

Erlang Solution:

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

Racket Solution:

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