Problem 1020: Number of Enclaves

Problem Information

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

Problem Description

You are given an

m x n

binary matrix

grid

, where

0

represents a sea cell and

1

represents a land cell.

A

move

consists of walking from one land cell to another adjacent (

4-directionally

) land cell or walking off the boundary of the

grid

.

Return

the number of land cells in

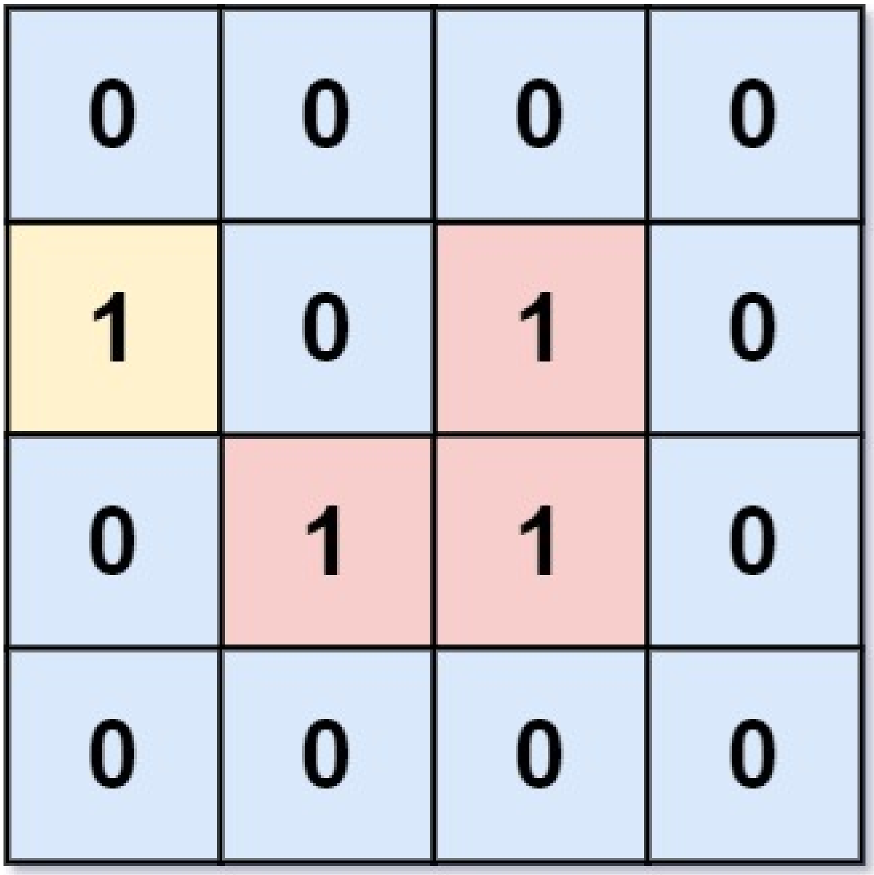
grid

for which we cannot walk off the boundary of the grid in any number of

moves

.

Example 1:



0	0	0	0
1	0	1	0
0	1	1	0
0	0	0	0

Input:

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

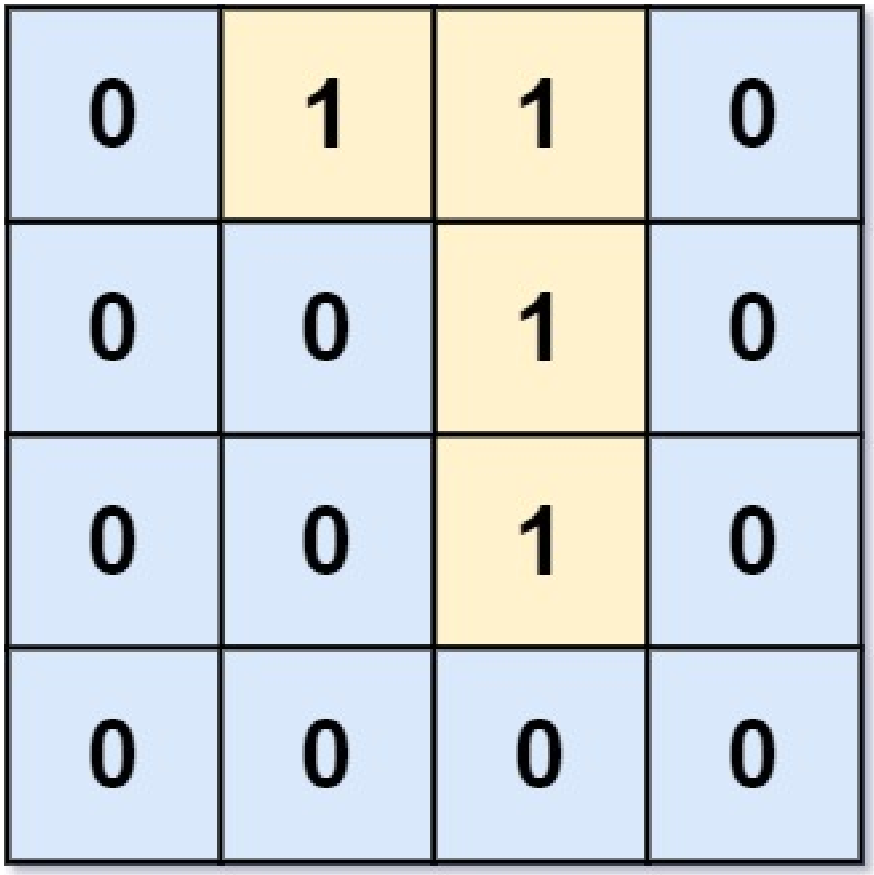
Output:

3

Explanation:

There are three 1s that are enclosed by 0s, and one 1 that is not enclosed because its on the boundary.

Example 2:



0	1	1	0
0	0	1	0
0	0	1	0
0	0	0	0

Input:

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

0

Explanation:

All 1s are either on the boundary or can reach the boundary.

Constraints:

m == grid.length

n == grid[i].length

1 <= m, n <= 500

grid[i][j]

is either

0

or

1

.

Code Snippets

C++:

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

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

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

}
}
```

C:

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

}
```

Go:

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

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

```
(define/contract (num-enclaves grid)
(-> (listof (listof exact-integer?)) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Number of Enclaves
* 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 numEnclaves(vector<vector<int>>& grid) {

}
};
```

Java Solution:

```
/**
* Problem: Number of Enclaves
* 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 numEnclaves(int[][] grid) {

}
}
```

Python3 Solution:

```
"""
Problem: Number of Enclaves
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 numEnclaves(self, grid: List[List[int]]) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Number of Enclaves
* 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 numEnclaves = function(grid) {

};
```

TypeScript Solution:

```
/**
* Problem: Number of Enclaves
* 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 numEnclaves(grid: number[][]): number {

};
```

C# Solution:

```
/*
* Problem: Number of Enclaves
* 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 NumEnclaves(int[][] grid) {

}
}
```

C Solution:

```
/*
* Problem: Number of Enclaves
* 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 numEnclaves(int** grid, int gridSize, int* gridColSize) {

}
```

Go Solution:

```
// Problem: Number of Enclaves
// 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 numEnclaves(grid [][]int) int {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Number of Enclaves
// 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 num_enclaves(grid: Vec<Vec<i32>>) -> i32 {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {
```

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

```
(define/contract (num-enclaves grid)
(-> (listof (listof exact-integer?)) exact-integer?)
)
```