Problem 1254: Number of Closed Islands

Problem Information

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

Problem Description

Given a 2D

grid

consists of

0s

(land) and

1s

(water). An

island

is a maximal 4-directionally connected group of

0

s

and a

closed island

is an island

totally

(all left, top, right, bottom) surrounded by

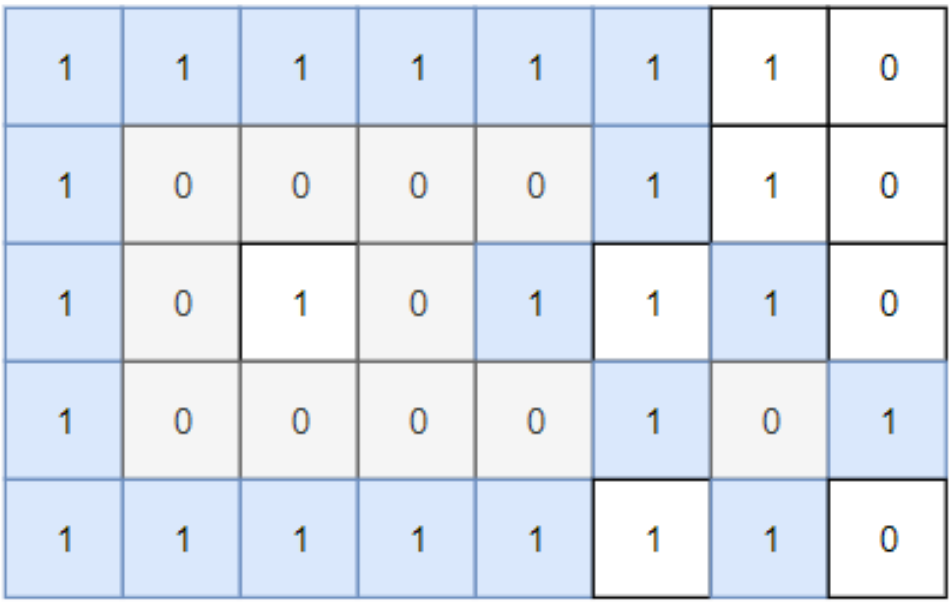
1s.

Return the number of

closed islands

.

Example 1:



Input:

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

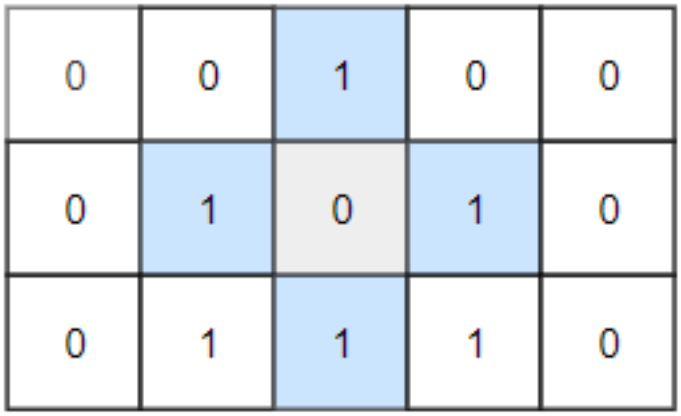
Output:

2

Explanation:

Islands in gray are closed because they are completely surrounded by water (group of 1s).

Example 2:



Input:

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

Output:

1

Example 3:

Input:

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

Output:

2

Constraints:

1 <= grid.length, grid[0].length <= 100

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

Code Snippets

C++:

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

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

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

}
}
```

C:

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

}
```

Go:

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

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
}
```

Scala:

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

}
```

```
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

```
(define/contract (closed-island grid)
(-> (listof (listof exact-integer?)) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Number of Closed Islands
* 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 closedIsland(vector<vector<int>>& grid) {
```

```
}
};
```

Java Solution:

```
/**
* Problem: Number of Closed Islands
* 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 closedIsland(int[][] grid) {

}
}
```

Python3 Solution:

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

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Number of Closed Islands
* 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 closedIsland = function(grid) {

};
```

TypeScript Solution:

```
/**
* Problem: Number of Closed Islands
* 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 closedIsland(grid: number[][]): number {

};
```

C# Solution:

```
/*
* Problem: Number of Closed Islands
* 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 ClosedIsland(int[][] grid) {

}
}
```

C Solution:

```
/*
* Problem: Number of Closed Islands
* 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 closedIsland(int** grid, int gridSize, int* gridColSize) {

}
```

Go Solution:

```
// Problem: Number of Closed Islands
// 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 closedIsland(grid [][]int) int {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Number of Closed Islands
// 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 closed_island(grid: Vec<Vec<i32>>) -> i32 {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
```

```
end
```

Erlang Solution:

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

Racket Solution:

```
(define/contract (closed-island grid)
(-> (listof (listof exact-integer?)) exact-integer?)
)
```