Problem 1568: Minimum Number of Days to Disconnect Island

Problem Information

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

Problem Description

You are given an

m x n

binary grid

grid

where

1

represents land and

0

represents water. An

island

is a maximal

4-directionally

(horizontal or vertical) connected group of

1

's.

The grid is said to be

connected

if we have

exactly one island

, otherwise is said

disconnected

.

In one day, we are allowed to change

any

single land cell

(1)

into a water cell

(0)

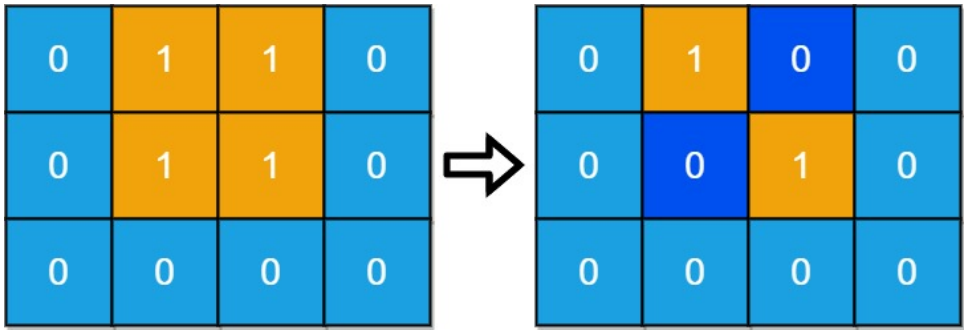
.

Return

the minimum number of days to disconnect the grid

.

Example 1:



Input:

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

Output:

2

Explanation:

We need at least 2 days to get a disconnected grid. Change land grid[1][1] and grid[0][2] to water and get 2 disconnected island.

Example 2:

Input:

grid = [[1,1]]

Output:

2

Explanation:

Grid of full water is also disconnected ([[1,1]] -> [[0,0]]), 0 islands.

Constraints:

m == grid.length

n == grid[i].length

1 <= m, n <= 30

grid[i][j]

is either

0

or

1

.

Code Snippets

C++:

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

}
};
```

Java:

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

}
```

```
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

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

}
}
```

C:

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

}
```

Go:

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

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

```
(define/contract (min-days grid)
(-> (listof (listof exact-integer?)) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Minimum Number of Days to Disconnect Island
* Difficulty: Hard
* 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 minDays(vector<vector<int>>& grid) {

}
};
```

Java Solution:

```
/**
* Problem: Minimum Number of Days to Disconnect Island
* Difficulty: Hard
* 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 minDays(int[][] grid) {

}
```

```
}
```

Python3 Solution:

```
"""
Problem: Minimum Number of Days to Disconnect Island
Difficulty: Hard
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 minDays(self, grid: List[List[int]]) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Minimum Number of Days to Disconnect Island
* Difficulty: Hard
* 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 minDays = function(grid) {

};
```

TypeScript Solution:

```
/**
* Problem: Minimum Number of Days to Disconnect Island
* Difficulty: Hard
* 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 minDays(grid: number[][]): number {

};
```

C# Solution:

```
/*
* Problem: Minimum Number of Days to Disconnect Island
* Difficulty: Hard
* 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 MinDays(int[][] grid) {

}
}
```

C Solution:

```
/*
* Problem: Minimum Number of Days to Disconnect Island
* Difficulty: Hard
* 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 minDays(int** grid, int gridSize, int* gridColSize) {

}
```

Go Solution:

```
// Problem: Minimum Number of Days to Disconnect Island
// Difficulty: Hard
// 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 minDays(grid [][]int) int {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

```
}
}
```

Rust Solution:

```
// Problem: Minimum Number of Days to Disconnect Island
// Difficulty: Hard
// 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 min_days(grid: Vec<Vec<i32>>) -> i32 {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

```
(define/contract (min-days grid)
(-> (listof (listof exact-integer?)) exact-integer?)
)
```