Problem 1905: Count Sub Islands

Problem Information

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

Problem Description

You are given two

m x n

binary matrices

grid1

and

grid2

containing only

0

's (representing water) and

1

's (representing land). An

island

is a group of

1

's connected

4-directionally

(horizontal or vertical). Any cells outside of the grid are considered water cells.

An island in

grid2

is considered a

sub-island

if there is an island in

grid1

that contains

all

the cells that make up

this

island in

grid2

.

Return the

number

of islands in

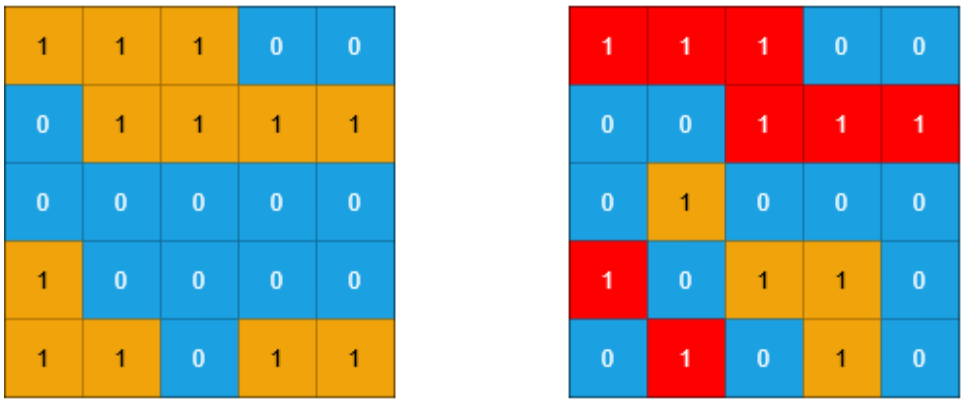
grid2

that are considered

sub-islands

.

Example 1:



Input:

grid1 = [[1,1,1,0,0],[0,1,1,1,1],[0,0,0,0,0],[1,0,0,0,0],[1,1,0,1,1]], grid2 = [[1,1,1,0,0],[0,0,1,1,1],[0,1,0,0,0],[1,0,1,1,0],[0,1,0,1,0]]

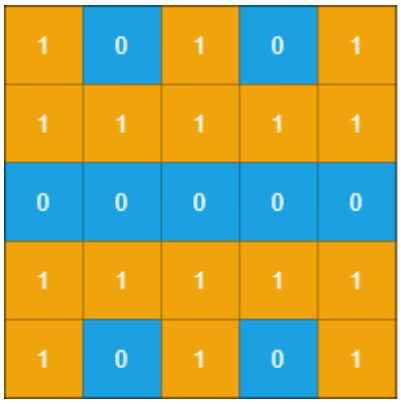
Output:

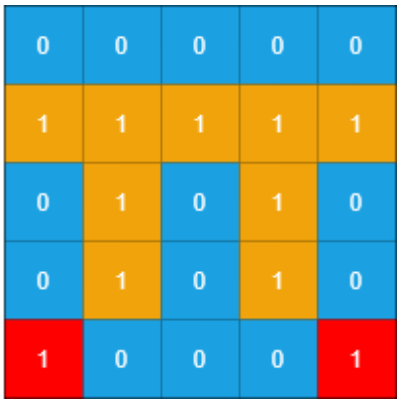
3

Explanation:

In the picture above, the grid on the left is grid1 and the grid on the right is grid2. The 1s colored red in grid2 are those considered to be part of a sub-island. There are three sub-islands.

Example 2:





Input:

grid1 = [[1,0,1,0,1],[1,1,1,1,1],[0,0,0,0,0],[1,1,1,1,1],[1,0,1,0,1]], grid2 = [[0,0,0,0,0],[1,1,1,1,1],[0,1,0,1,0],[0,1,0,1,0],[1,0,0,0,1]]

Output:

2

Explanation:

In the picture above, the grid on the left is grid1 and the grid on the right is grid2. The 1s colored red in grid2 are those considered to be part of a sub-island. There are two sub-islands.

Constraints:

m == grid1.length == grid2.length

n == grid1[i].length == grid2[i].length

1 <= m, n <= 500

grid1[i][j]

and

grid2[i][j]

are either

0

or

1

.

Code Snippets

C++:

```
class Solution {
public:
int countSubIslands(vector<vector<int>>& grid1, vector<vector<int>>& grid2) {

}
};
```

Java:

```
class Solution {
public int countSubIslands(int[][] grid1, int[][] grid2) {

}
}
```

Python3:

```
class Solution:
def countSubIslands(self, grid1: List[List[int]], grid2: List[List[int]]) ->
int:
```

Python:

```
class Solution(object):
def countSubIslands(self, grid1, grid2):
"""
:type grid1: List[List[int]]
:type grid2: List[List[int]]
```

```
:rtype: int
"""
```

JavaScript:

```
/**
* @param {number[][]} grid1
* @param {number[][]} grid2
* @return {number}
*/
var countSubIslands = function(grid1, grid2) {

};
```

TypeScript:

```
function countSubIslands(grid1: number[][], grid2: number[][]): number {

};
```

C#:

```
public class Solution {
public int CountSubIslands(int[][] grid1, int[][] grid2) {

}
}
```

C:

```
int countSubIslands(int** grid1, int grid1Size, int* grid1ColSize, int**
grid2, int grid2Size, int* grid2ColSize) {

}
```

Go:

```
func countSubIslands(grid1 [][]int, grid2 [][]int) int {

}
```

Kotlin:

```
class Solution {
fun countSubIslands(grid1: Array<IntArray>, grid2: Array<IntArray>): Int {

}
}
```

Swift:

```
class Solution {
func countSubIslands(_ grid1: [[Int]], _ grid2: [[Int]]) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn count_sub_islands(grid1: Vec<Vec<i32>>, grid2: Vec<Vec<i32>>) -> i32 {

}
}
```

Ruby:

```
# @param {Integer[][]} grid1
# @param {Integer[][]} grid2
# @return {Integer}
def count_sub_islands(grid1, grid2)

end
```

PHP:

```
class Solution {

/**
* @param Integer[][] $grid1
* @param Integer[][] $grid2
* @return Integer
*/
function countSubIslands($grid1, $grid2) {

}
```

```
}
```

Dart:

```
class Solution {
int countSubIslands(List<List<int>> grid1, List<List<int>> grid2) {

}
}
```

Scala:

```
object Solution {
def countSubIslands(grid1: Array[Array[Int]], grid2: Array[Array[Int]]): Int
= {

}
}
```

Elixir:

```
defmodule Solution do
@spec count_sub_islands(grid1 :: [[integer]], grid2 :: [[integer]]) ::
integer
def count_sub_islands(grid1, grid2) do

end
end
```

Erlang:

```
-spec count_sub_islands(Grid1 :: [[integer()]], Grid2 :: [[integer()]]) ->
integer().
count_sub_islands(Grid1, Grid2) ->
.
```

Racket:

```
(define/contract (count-sub-islands grid1 grid2)
(-> (listof (listof exact-integer?)) (listof (listof exact-integer?))
exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Count Sub 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 countSubIslands(vector<vector<int>>& grid1, vector<vector<int>>& grid2) {

}
};
```

Java Solution:

```
/**
* Problem: Count Sub 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 countSubIslands(int[][] grid1, int[][] grid2) {

}
}
```

Python3 Solution:

```
"""
Problem: Count Sub 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 countSubIslands(self, grid1: List[List[int]], grid2: List[List[int]]) ->
int:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def countSubIslands(self, grid1, grid2):
"""
:type grid1: List[List[int]]
:type grid2: List[List[int]]
:rtype: int
"""
```

JavaScript Solution:

```
/**
* Problem: Count Sub 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[][]} grid1
* @param {number[][]} grid2
* @return {number}
```

```
*/
var countSubIslands = function(grid1, grid2) {

};
```

TypeScript Solution:

```
/**
* Problem: Count Sub 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 countSubIslands(grid1: number[][], grid2: number[][]): number {

};
```

C# Solution:

```
/*
* Problem: Count Sub 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 CountSubIslands(int[][] grid1, int[][] grid2) {

}
}
```

C Solution:

```
/*
* Problem: Count Sub 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 countSubIslands(int** grid1, int grid1Size, int* grid1ColSize, int**
grid2, int grid2Size, int* grid2ColSize) {

}
```

Go Solution:

```
// Problem: Count Sub 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 countSubIslands(grid1 [][]int, grid2 [][]int) int {

}
```

Kotlin Solution:

```
class Solution {
fun countSubIslands(grid1: Array<IntArray>, grid2: Array<IntArray>): Int {

}
}
```

Swift Solution:

```
class Solution {
func countSubIslands(_ grid1: [[Int]], _ grid2: [[Int]]) -> Int {
```

```
}
}
```

Rust Solution:

```
// Problem: Count Sub 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 count_sub_islands(grid1: Vec<Vec<i32>>, grid2: Vec<Vec<i32>>) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer[][]} grid1
# @param {Integer[][]} grid2
# @return {Integer}
def count_sub_islands(grid1, grid2)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[][] $grid1
* @param Integer[][] $grid2
* @return Integer
*/
function countSubIslands($grid1, $grid2) {

}
}
```

Dart Solution:

```
class Solution {
int countSubIslands(List<List<int>> grid1, List<List<int>> grid2) {

}
}
```

Scala Solution:

```
object Solution {
def countSubIslands(grid1: Array[Array[Int]], grid2: Array[Array[Int]]): Int
= {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec count_sub_islands(grid1 :: [[integer]], grid2 :: [[integer]]) ::
integer
def count_sub_islands(grid1, grid2) do

end
end
```

Erlang Solution:

```
-spec count_sub_islands(Grid1 :: [[integer()]], Grid2 :: [[integer()]]) ->
integer().
count_sub_islands(Grid1, Grid2) ->
.
```

Racket Solution:

```
(define/contract (count-sub-islands grid1 grid2)
(-> (listof (listof exact-integer?)) (listof (listof exact-integer?))
exact-integer?)
)
```