Problem 694: Number of Distinct Islands

Problem Information

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

Problem Description

You are given an

m x n

binary matrix

grid

. An island is a group of

1

's (representing land) connected

4-directionally

(horizontal or vertical.) You may assume all four edges of the grid are surrounded by water.

An island is considered to be the same as another if and only if one island can be translated (and not rotated or reflected) to equal the other.

Return

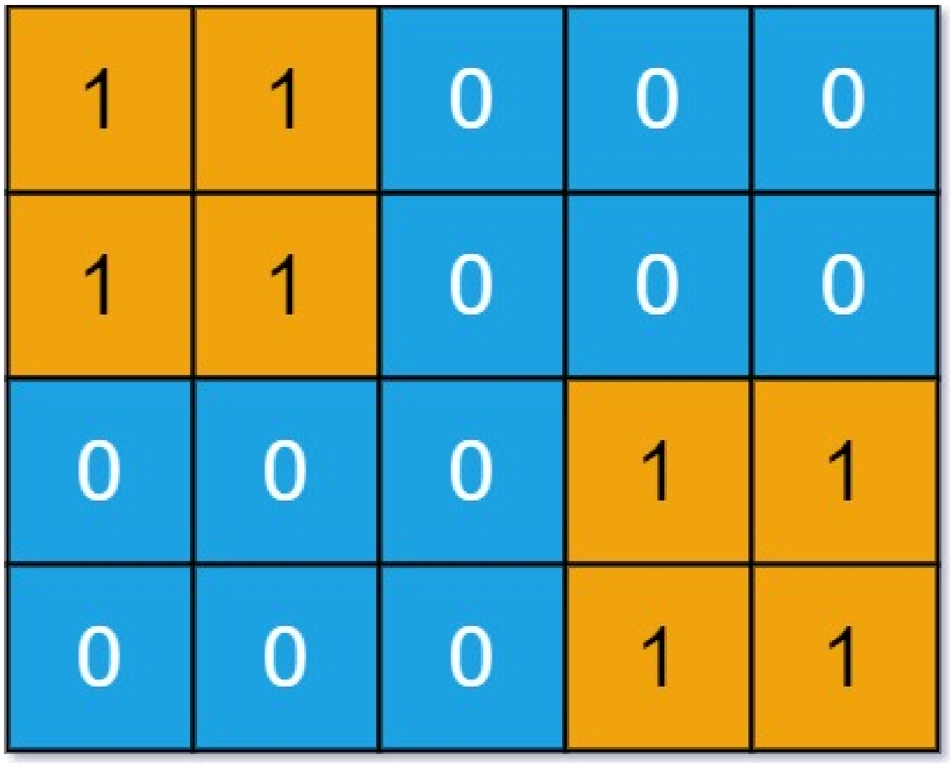
the number of

distinct

islands

.

Example 1:



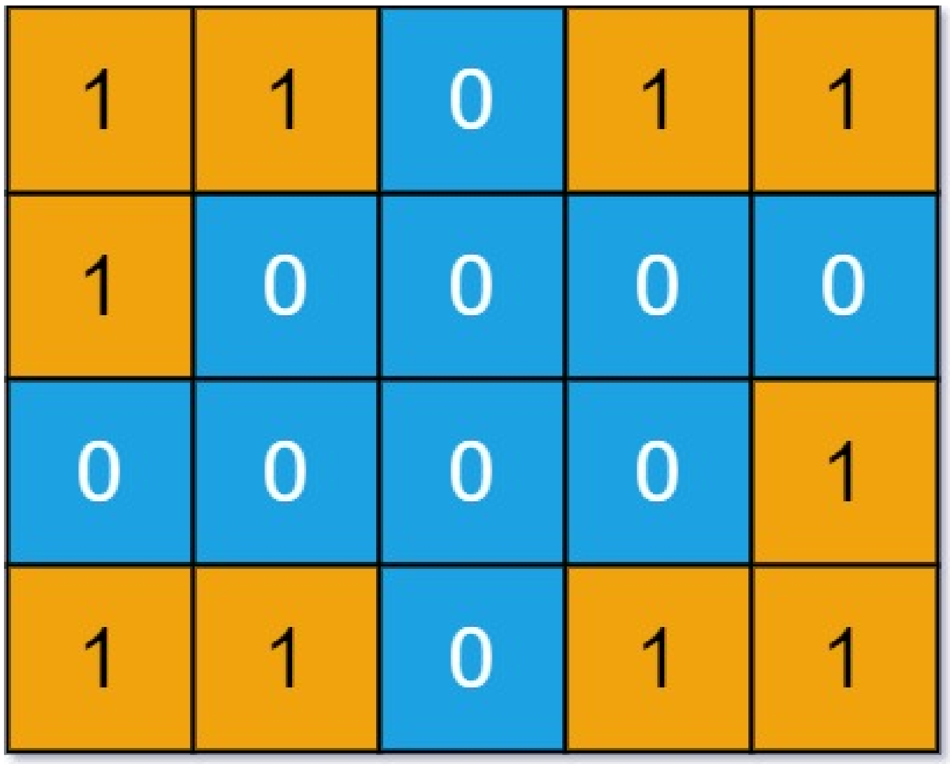
Input:

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

Output:

1

Example 2:



Input:

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

Output:

3

Constraints:

m == grid.length

n == grid[i].length

1 <= m, n <= 50

grid[i][j]

is either

0

or

1

.

Code Snippets

C++:

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

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

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

}
}
```

C:

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

}
```

Go:

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

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

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

Solutions

C++ Solution:

```
/*
* Problem: Number of Distinct Islands
* Difficulty: Medium
* Tags: graph, hash, search
*
* Approach: Use hash map for O(1) lookups
* Time Complexity: O(n) to O(n^2) depending on approach
* Space Complexity: O(n) for hash map
*/
```

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

}
};
```

Java Solution:

```
/**
* Problem: Number of Distinct Islands
* Difficulty: Medium
* Tags: graph, hash, search
*
* Approach: Use hash map for O(1) lookups
* Time Complexity: O(n) to O(n^2) depending on approach
* Space Complexity: O(n) for hash map
*/

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

}
}
```

Python3 Solution:

```
"""
Problem: Number of Distinct Islands
Difficulty: Medium
Tags: graph, hash, search

Approach: Use hash map for O(1) lookups
Time Complexity: O(n) to O(n^2) depending on approach
Space Complexity: O(n) for hash map
"""

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

```
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Number of Distinct Islands
* Difficulty: Medium
* Tags: graph, hash, search
*
* Approach: Use hash map for O(1) lookups
* Time Complexity: O(n) to O(n^2) depending on approach
* Space Complexity: O(n) for hash map
*/

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

};
```

TypeScript Solution:

```
/**
* Problem: Number of Distinct Islands
* Difficulty: Medium
* Tags: graph, hash, search
*
* Approach: Use hash map for O(1) lookups
* Time Complexity: O(n) to O(n^2) depending on approach
* Space Complexity: O(n) for hash map
```

```
*/

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

};
```

C# Solution:

```
/*
* Problem: Number of Distinct Islands
* Difficulty: Medium
* Tags: graph, hash, search
*
* Approach: Use hash map for O(1) lookups
* Time Complexity: O(n) to O(n^2) depending on approach
* Space Complexity: O(n) for hash map
*/

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

}
}
```

C Solution:

```
/*
* Problem: Number of Distinct Islands
* Difficulty: Medium
* Tags: graph, hash, search
*
* Approach: Use hash map for O(1) lookups
* Time Complexity: O(n) to O(n^2) depending on approach
* Space Complexity: O(n) for hash map
*/

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

}
```

Go Solution:

```
// Problem: Number of Distinct Islands
// Difficulty: Medium
// Tags: graph, hash, search
//
// Approach: Use hash map for O(1) lookups
// Time Complexity: O(n) to O(n^2) depending on approach
// Space Complexity: O(n) for hash map

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

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Number of Distinct Islands
// Difficulty: Medium
// Tags: graph, hash, search
//
// Approach: Use hash map for O(1) lookups
// Time Complexity: O(n) to O(n^2) depending on approach
// Space Complexity: O(n) for hash map

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

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

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