Problem 200: Number of Islands

Problem Information

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

Problem Description

Given an

m x n

2D binary grid

grid

which represents a map of

'1'

s (land) and

'0'

s (water), return

the number of islands

.

An

island

is surrounded by water and is formed by connecting adjacent lands horizontally or vertically. You may assume all four edges of the grid are all surrounded by water.

Example 1:

Input:

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

Output:

1

Example 2:

Input:

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

Output:

3

Constraints:

m == grid.length

n == grid[i].length

1 <= m, n <= 300

grid[i][j]

is

'0'

or

'1'

.

Code Snippets

C++:

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

}
};
```

Java:

```
class Solution {
public int numIslands(char[][] grid) {

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

```
function numIslands(grid: string[][]): number {

};
```

C#:

```
public class Solution {
public int NumIslands(char[][] grid) {

}
}
```

C:

```
int numIslands(char** grid, int gridSize, int* gridColSize) {

}
```

Go:

```
func numIslands(grid [][]byte) int {

}
```

Kotlin:

```
class Solution {
fun numIslands(grid: Array<CharArray>): Int {

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

```
class Solution {
int numIslands(List<List<String>> grid) {

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

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

Solutions

C++ Solution:

```
/*
* Problem: Number of 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 numIslands(vector<vector<char>>& grid) {

}
};
```

Java Solution:

```
/**
* Problem: Number of 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 numIslands(char[][] grid) {

}
}
```

Python3 Solution:

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

```
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Number of 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 {character[][]} grid
* @return {number}
*/
var numIslands = function(grid) {

};
```

TypeScript Solution:

```
/**
* Problem: Number of 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 numIslands(grid: string[][]): number {

};
```

C# Solution:

```
/*
* Problem: Number of 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 NumIslands(char[][] grid) {

}
}
```

C Solution:

```
/*
* Problem: Number of 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 numIslands(char** grid, int gridSize, int* gridColSize) {

}
```

Go Solution:

```
// Problem: Number of 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 numIslands(grid [][]byte) int {

}
```

Kotlin Solution:

```
class Solution {
fun numIslands(grid: Array<CharArray>): Int {

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Number of 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 num_islands(grid: Vec<Vec<char>>) -> i32 {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

```
class Solution {
int numIslands(List<List<String>> grid) {

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

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