Problem 695: Max Area of Island

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.

The

area

of an island is the number of cells with a value

1

in the island.

Return

the maximum

area

of an island in

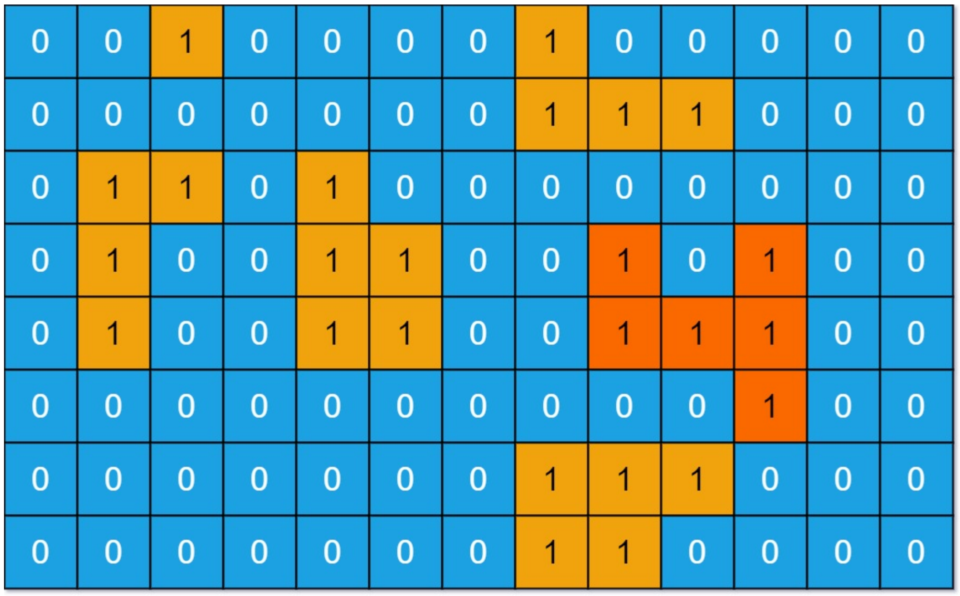
grid

. If there is no island, return

0

.

Example 1:



0	0	1	0	0	0	0	1	0	0	0	0	0
0	0	0	0	0	0	0	1	1	1	0	0	0
0	1	1	0	1	0	0	0	0	0	0	0	0
0	1	0	0	1	1	0	0	1	0	1	0	0
0	1	0	0	1	1	0	0	1	1	1	0	0
0	0	0	0	0	0	0	0	0	0	1	0	0
0	0	0	0	0	0	0	1	1	1	0	0	0
0	0	0	0	0	0	0	1	1	0	0	0	0

Input:

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

Output:

6

Explanation:

The answer is not 11, because the island must be connected 4-directionally.

Example 2:

Input:

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

Output:

0

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 maxAreaOfIsland(vector<vector<int>>& grid) {

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

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

}
}
```

C:

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

}
```

Go:

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

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
}
```

Scala:

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

}

}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

```
(define/contract (max-area-of-island grid)
(-> (listof (listof exact-integer?)) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Max Area of Island
* 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 maxAreaOfIsland(vector<vector<int>>& grid) {
}
};
```

Java Solution:

```
/**
* Problem: Max Area of Island
* 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 maxAreaOfIsland(int[][] grid) {

}
}
```

Python3 Solution:

```
"""
Problem: Max Area of Island
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 maxAreaOfIsland(self, grid: List[List[int]]) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Max Area of Island
* 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 maxAreaOfIsland = function(grid) {

};
```

TypeScript Solution:

```
/**
* Problem: Max Area of Island
* 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 maxAreaOfIsland(grid: number[][]): number {

};
```

C# Solution:

```
/*
* Problem: Max Area of Island
* 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 MaxAreaOfIsland(int[][] grid) {

}
}
```

C Solution:

```
/*
* Problem: Max Area of Island
* 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 maxAreaOfIsland(int** grid, int gridSize, int* gridColSize) {

}
```

Go Solution:

```
// Problem: Max Area of Island
// 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 maxAreaOfIsland(grid [][]int) int {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Max Area of Island
// 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 max_area_of_island(grid: Vec<Vec<i32>>) -> i32 {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
```

```

end
```

Erlang Solution:

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

Racket Solution:

```
(define/contract (max-area-of-island grid)
(-> (listof (listof exact-integer?)) exact-integer?)
)
```