Problem 2658: Maximum Number of Fish in a Grid

Problem Information

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

Problem Description

You are given a

0-indexed

2D matrix

grid

of size

m x n

, where

(r, c)

represents:

A

land

cell if

grid[r][c] = 0

, or

A

water

cell containing

grid[r][c]

fish, if

grid[r][c] > 0

.

A fisher can start at any

water

cell

(r, c)

and can do the following operations any number of times:

Catch all the fish at cell

(r, c)

, or

Move to any adjacent

water

cell.

Return

the

maximum

number of fish the fisher can catch if he chooses his starting cell optimally, or

0

if no water cell exists.

An

adjacent

cell of the cell

(r, c)

, is one of the cells

(r, c + 1)

,

(r, c - 1)

,

(r + 1, c)

or

(r - 1, c)

if it exists.

Example 1:

0	2	1	0
4	0	0	3
1	0	0	4
0	3	2	0

Input:

grid = [[0,2,1,0],[4,0,0,3],[1,0,0,4],[0,3,2,0]]

Output:

7

Explanation:

The fisher can start at cell

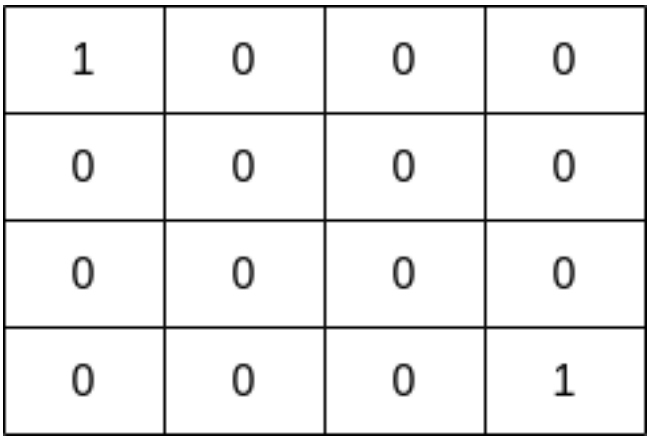
(1,3)

and collect 3 fish, then move to cell

(2,3)

and collect 4 fish.

Example 2:



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

Input:

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

Output:

1

Explanation:

The fisher can start at cells (0,0) or (3,3) and collect a single fish.

Constraints:

m == grid.length

n == grid[i].length

1 <= m, n <= 10

0 <= grid[i][j] <= 10

Code Snippets

C++:

```
class Solution {
public:
```

```
int findMaxFish(vector<vector<int>>& grid) {

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

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

}
}
```

C:

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

}
```

Go:

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

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

```
end
end
```

Erlang:

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

Racket:

```
(define/contract (find-max-fish grid)
(-> (listof (listof exact-integer?)) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Maximum Number of Fish in a Grid
* 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 findMaxFish(vector<vector<int>>& grid) {

}
};
```

Java Solution:

```
/**
* Problem: Maximum Number of Fish in a Grid
```

```
* 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 findMaxFish(int[][] grid) {

}
}
```

Python3 Solution:

```
"""
Problem: Maximum Number of Fish in a Grid
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 findMaxFish(self, grid: List[List[int]]) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Maximum Number of Fish in a Grid
* 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 findMaxFish = function(grid) {

};
```

TypeScript Solution:

```
/**
* Problem: Maximum Number of Fish in a Grid
* 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 findMaxFish(grid: number[][]): number {

};
```

C# Solution:

```
/*
* Problem: Maximum Number of Fish in a Grid
* 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 FindMaxFish(int[][] grid) {

}
}
```

C Solution:

```
/*
* Problem: Maximum Number of Fish in a Grid
* 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 findMaxFish(int** grid, int gridSize, int* gridColSize) {

}
```

Go Solution:

```
// Problem: Maximum Number of Fish in a Grid
// 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 findMaxFish(grid [][]int) int {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Maximum Number of Fish in a Grid
// 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 find_max_fish(grid: Vec<Vec<i32>>) -> i32 {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {
```

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

```
(define/contract (find-max-fish grid)
(-> (listof (listof exact-integer?)) exact-integer?)
)
```