Problem 1162: As Far from Land as Possible

Problem Information

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

Problem Description

Given an

n x n

grid

containing only values

0

and

1

, where

0

represents water and

1

represents land, find a water cell such that its distance to the nearest land cell is maximized, and return the distance. If no land or water exists in the grid, return

-1

.

The distance used in this problem is the Manhattan distance: the distance between two cells

(x0, y0)

and

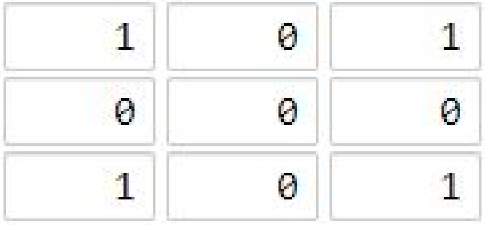
(x1, y1)

is

|x0 - x1| + |y0 - y1|

.

Example 1:



1	0	1
0	0	0
1	0	1

Input:

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

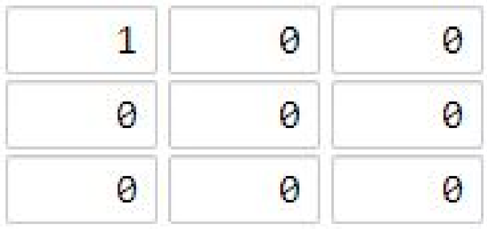
Output:

2

Explanation:

The cell (1, 1) is as far as possible from all the land with distance 2.

Example 2:



1	0	0
0	0	0
0	0	0

Input:

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

Output:

4

Explanation:

The cell (2, 2) is as far as possible from all the land with distance 4.

Constraints:

n == grid.length

n == grid[i].length

1 <= n <= 100

grid[i][j]

is

0

or

1

Code Snippets

C++:

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

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

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

}
}
```

C:

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

}
```

Go:

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

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

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

Solutions

C++ Solution:

```
/*
* Problem: As Far from Land as Possible
* Difficulty: Medium
* Tags: array, dp, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

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

}
};
```

Java Solution:

```
/**
* Problem: As Far from Land as Possible
* Difficulty: Medium
* Tags: array, dp, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

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

}
}
```

Python3 Solution:

```
"""
Problem: As Far from Land as Possible
Difficulty: Medium
Tags: array, dp, search

Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(n) or O(n * m) for DP table
"""

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

Python Solution:

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

```
"""
```

JavaScript Solution:

```
/**
* Problem: As Far from Land as Possible
* Difficulty: Medium
* Tags: array, dp, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

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

};
```

TypeScript Solution:

```
/**
* Problem: As Far from Land as Possible
* Difficulty: Medium
* Tags: array, dp, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

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

};
```

C# Solution:

```
/*
* Problem: As Far from Land as Possible
* Difficulty: Medium
* Tags: array, dp, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

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

}
}
```

C Solution:

```
/*
* Problem: As Far from Land as Possible
* Difficulty: Medium
* Tags: array, dp, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

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

}
```

Go Solution:

```
// Problem: As Far from Land as Possible
// Difficulty: Medium
// Tags: array, dp, search
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) or O(n * m) for DP table
```

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

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: As Far from Land as Possible
// Difficulty: Medium
// Tags: array, dp, search
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) or O(n * m) for DP table

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

}
}
```

Ruby Solution:

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

```
end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

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