Problem 2088: Count Fertile Pyramids in a Land

Problem Information

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

Problem Description

A farmer has a

rectangular grid

of land with

m

rows and

n

columns that can be divided into unit cells. Each cell is either

fertile

(represented by a

1

) or

barren

(represented by a

0

). All cells outside the grid are considered barren.

A

pyramidal plot

of land can be defined as a set of cells with the following criteria:

The number of cells in the set has to be

greater than

1

and all cells must be

fertile

.

The

apex

of a pyramid is the

topmost

cell of the pyramid. The

height

of a pyramid is the number of rows it covers. Let

(r, c)

be the apex of the pyramid, and its height be h

. Then, the plot comprises of cells

(i, j)

where

r <= i <= r + h - 1

and

c - (i - r) <= j <= c + (i - r)

.

An

inverse pyramidal plot

of land can be defined as a set of cells with similar criteria:

The number of cells in the set has to be

greater than

1

and all cells must be

fertile

.

The

apex

of an inverse pyramid is the

bottommost

cell of the inverse pyramid. The

height

of an inverse pyramid is the number of rows it covers. Let

(r, c)

be the apex of the pyramid, and its height be

h

. Then, the plot comprises of cells

(i, j)

where

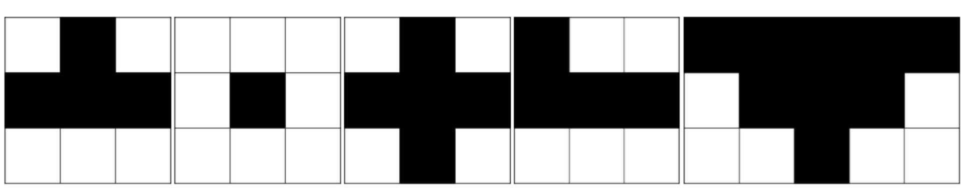
r - h + 1 <= i <= r

and

c - (r - i) <= j <= c + (r - i)

.

Some examples of valid and invalid pyramidal (and inverse pyramidal) plots are shown below. Black cells indicate fertile cells.



valid invalid invalid invalid valid

Given a

0-indexed

m x n

binary matrix

grid

representing the farmland, return

the

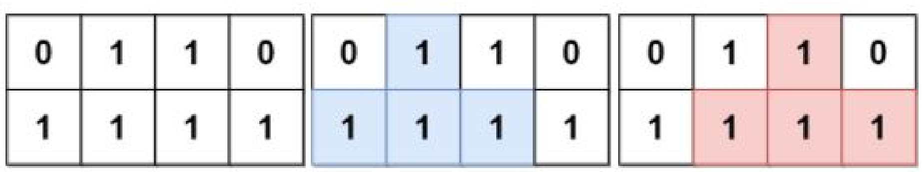
total number

of pyramidal and inverse pyramidal plots that can be found in

grid

.

Example 1:



Input:

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

Output:

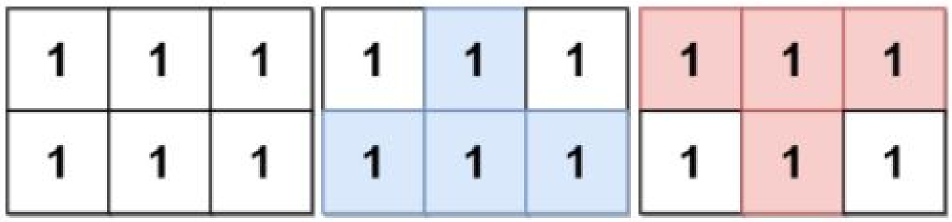
2

Explanation:

The 2 possible pyramidal plots are shown in blue and red respectively. There are no inverse pyramidal plots in this grid. Hence total number of pyramidal and inverse pyramidal plots is 2

+ 0 = 2.

Example 2:



Input:

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

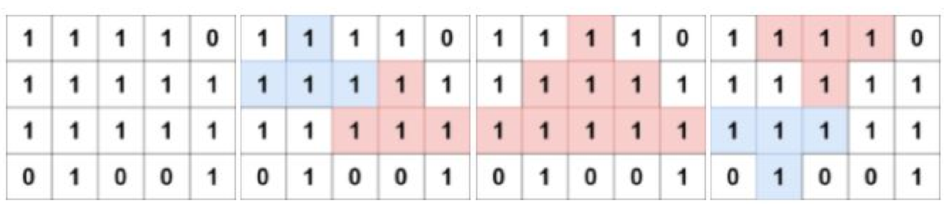
Output:

2

Explanation:

The pyramidal plot is shown in blue, and the inverse pyramidal plot is shown in red. Hence the total number of plots is 1 + 1 = 2.

Example 3:



Input:

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

Output:

13

Explanation:

There are 7 pyramidal plots, 3 of which are shown in the 2nd and 3rd figures. There are 6 inverse pyramidal plots, 2 of which are shown in the last figure. The total number of plots is 7 + 6 = 13.

Constraints:

m == grid.length

n == grid[i].length

1 <= m, n <= 1000

1 <= m * n <= 10

5

grid[i][j]

is either

0

or

1

.

Code Snippets

C++:

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

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

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

```


}
}
```

C:

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

}
```

Go:

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

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

```
(define/contract (count-pyramids grid)
(-> (listof (listof exact-integer?)) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Count Fertile Pyramids in a Land
* Difficulty: Hard
* Tags: array, dp
*
* 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 countPyramids(vector<vector<int>>& grid) {

}
};
```

Java Solution:

```
/**
* Problem: Count Fertile Pyramids in a Land
* Difficulty: Hard
* Tags: array, dp
*
* 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 countPyramids(int[][] grid) {

}
}
```

Python3 Solution:

```
"""
Problem: Count Fertile Pyramids in a Land
Difficulty: Hard
Tags: array, dp

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 countPyramids(self, grid: List[List[int]]) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Count Fertile Pyramids in a Land
* Difficulty: Hard
```

```
* Tags: array, dp
*
* 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 countPyramids = function(grid) {

};
```

TypeScript Solution:

```
/**
* Problem: Count Fertile Pyramids in a Land
* Difficulty: Hard
* Tags: array, dp
*
* 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 countPyramids(grid: number[][]): number {

};
```

C# Solution:

```
/*
* Problem: Count Fertile Pyramids in a Land
* Difficulty: Hard
* Tags: array, dp
*
* 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 CountPyramids(int[][] grid) {

}
}
```

C Solution:

```
/*
* Problem: Count Fertile Pyramids in a Land
* Difficulty: Hard
* Tags: array, dp
*
* 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 countPyramids(int** grid, int gridSize, int* gridColSize) {

}
```

Go Solution:

```
// Problem: Count Fertile Pyramids in a Land
// Difficulty: Hard
// Tags: array, dp
//
// 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 countPyramids(grid [][]int) int {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Count Fertile Pyramids in a Land
// Difficulty: Hard
// Tags: array, dp
//
// 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 count_pyramids(grid: Vec<Vec<i32>>) -> i32 {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {
```

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

```
(define/contract (count-pyramids grid)
(-> (listof (listof exact-integer?)) exact-integer?)
)
```