Problem 463: Island Perimeter

Problem Information

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

Problem Description

You are given

row x col

grid

representing a map where

grid[i][j] = 1

represents land and

grid[i][j] = 0

represents water.

Grid cells are connected

horizontally/vertically

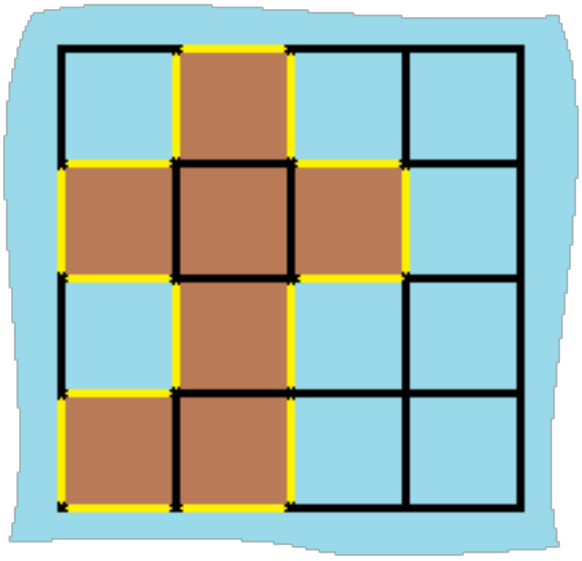
(not diagonally). The

grid

is completely surrounded by water, and there is exactly one island (i.e., one or more connected land cells).

The island doesn't have "lakes", meaning the water inside isn't connected to the water around the island. One cell is a square with side length 1. The grid is rectangular, width and height don't exceed 100. Determine the perimeter of the island.

Example 1:



Input:

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

Output:

16

Explanation:

The perimeter is the 16 yellow stripes in the image above.

Example 2:

Input:

grid = [[1]]

Output:

4

Example 3:

Input:

grid = [[1,0]]

Output:

4

Constraints:

row == grid.length

col == grid[i].length

1 <= row, col <= 100

grid[i][j]

is

0

or

1

.

There is exactly one island in

grid

.

Code Snippets

C++:

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

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

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

}
}
```

C:

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

}
```

Go:

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

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

```
(define/contract (island-perimeter grid)
(-> (listof (listof exact-integer?)) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Island Perimeter
* Difficulty: Easy
* Tags: array, 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 islandPerimeter(vector<vector<int>>& grid) {

}
};
```

Java Solution:

```
/**
* Problem: Island Perimeter
* Difficulty: Easy
* Tags: array, 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 islandPerimeter(int[][] grid) {

}
}
```

Python3 Solution:

```
"""
Problem: Island Perimeter
Difficulty: Easy
Tags: array, 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 islandPerimeter(self, grid: List[List[int]]) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

```
"""
```

JavaScript Solution:

```
/**
* Problem: Island Perimeter
* Difficulty: Easy
* Tags: array, 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 islandPerimeter = function(grid) {

};
```

TypeScript Solution:

```
/**
* Problem: Island Perimeter
* Difficulty: Easy
* Tags: array, 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 islandPerimeter(grid: number[][]): number {

};
```

C# Solution:

```
/*
* Problem: Island Perimeter
* Difficulty: Easy
* Tags: array, 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 IslandPerimeter(int[][] grid) {

}
}
```

C Solution:

```
/*
* Problem: Island Perimeter
* Difficulty: Easy
* Tags: array, 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 islandPerimeter(int** grid, int gridSize, int* gridColSize) {

}
```

Go Solution:

```
// Problem: Island Perimeter
// Difficulty: Easy
// Tags: array, 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 islandPerimeter(grid [][]int) int {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Island Perimeter
// Difficulty: Easy
// Tags: array, 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 island_perimeter(grid: Vec<Vec<i32>>) -> i32 {

}
}
```

Ruby Solution:

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

```
end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

```
(define/contract (island-perimeter grid)
(-> (listof (listof exact-integer?)) exact-integer?)
)
```