463. Island Perimeter

**Depth-First Search** 

# **Problem Description**

Easy

represents water. The key points about the map are:

In this problem, you are presented with a 2D grid that represents a map, where the value 1 indicates land and the value 0

• There's exactly one island, and it is made up of land cells (1's) that are connected horizontally or vertically. • The surrounding cells outside the grid are water (0 's).

Matrix

• There are no "lakes," meaning there are no enclosed areas of water within the island.

Array

- Each cell of the grid is a square with a side length of 1.
- The dimensions of the grid will not be larger than 100×100.
- Your task is to determine and return the perimeter of the island in this grid. Remember that the perimeter counts the boundary

**Breadth-First Search** 

To solve this problem, we need to calculate the total perimeter contributed by each land cell in the grid. Since we're working with a grid that has connected cells horizontally and vertically, each land cell that does not touch another land cell contributes 4 units to the perimeter (as it has four sides).

2. Iterate through each cell in the grid. 3. If a cell is land (1), increment the perimeter count by 4 (all possible sides of a single cell).

Here's the intuitive step-by-step approach:

that separates land cells from water cells.

- 4. Then, check the adjacent cells:

1. Initialize a perimeter count to 0.

o If there is a land cell to the right (horizontally adjacent), the shared edge does not contribute to the perimeter, so we subtract 2 from the perimeter count (as it removes one edge from each of the two adjacent land cells).

Similarly, if there is a land cell below (vertically adjacent), subtract 2 for the shared edge.

6. Return the total perimeter count. This approach works because it dynamically adjusts the perimeter count based on the land cell's adjacency with other land cells,

5. Continue this process for all land cells in the grid.

- ensuring that shared edges are only counted once.
- **Solution Approach**

• To check the cell below the current cell (if it exists and is a land cell), we perform:

To check the cell to the right of the current cell (if it exists and is a land cell), we perform:

from our total perimeter count -1 for each of the two cells sharing the edge.

if i < m - 1 and grid[i + 1][j] == 1:</pre>

if j < n - 1 and grid[i][j + 1] == 1:</pre>

The solution approach for determining the perimeter of the island adheres to the following algorithmic steps, aligning with the explained intuition:

Initiate a Counter for Perimeter: Start with a variable ans, initialized to 0, which will keep track of the island's perimeter.

Iterate over Grid Cells: Use a nested loop to go through every cell in the grid. Let m be the number of rows and n be the

### number of columns of the grid: for i in range(m):

for j in range(n):

ans -= 2

**Example Walkthrough** 

represents water:

1 1 0

0 1 0

Check for Land Cells: If the current cell grid[i][j] is a land cell (1), increment the perimeter counter by 4: if grid[i][j] == 1: ans += 4

- This is because a land cell has 4 potential edges contributing to the perimeter. Check for Adjacent Land: Determine if the land cell has adjacent land cells that would reduce the perimeter:
- ans -= 2
- We use the condition j < n 1 to ensure we're not on the right most column before checking the cell to the right. Subtract Shared Edges: The subtraction of 2 from the perimeter ans in the case of adjacent land cells accounts for the fact

Return Perimeter Count: After the entire grid has been processed, return the calculated perimeter ans.

We use the condition i < m - 1 to ensure we're not on the bottom most row before checking the cell below.

The algorithm makes use of nested loops to process a 2D matrix, while the main data structure utilized is the 2D list given as input. This approach is straightforward with a linear runtime that corresponds to the size of the grid (0(m\*n)), as each cell is visited exactly once.

Let's walk through a simple example to illustrate the solution approach. Consider a 3×3 grid, where 1 represents land and 0

that each shared edge is part of the perimeter of two adjacent cells. Since this edge cannot be counted twice, we subtract 2

Grid: 1 0 1

Following the steps outlined in the algorithm: **Initiate a Counter for Perimeter:** Start with ans = 0. Iterate over Grid Cells: We will examine each cell to determine if it contributes to the perimeter.

## • The first cell (1,0) is a 1 (land). We check the cell above (0,0) which is also a 1. This means we have adjacent land cells, so ans += 4 and

Second row:

edge)  $\Rightarrow$  ans = 10.

First row:

ans -= 2 for the shared edge  $\Rightarrow$  ans = 10. • The second cell (1,1) is a 1 (land). It's surrounded by land on two sides (above and to the left), so ans += 4 and ans -= 4 (2 for each shared

• The second cell (2,1) is a 1 (land). It's surrounded by land above only, so ans += 4 and ans -= 2 for the shared edge ⇒ ans = 12.

Check for Land Cells: We have done this part during our iteration and added 4 to ans for each land cell.

Third row: • The first cell (2,0) is a 0 (water), so no change to ans.

• The first cell (0,0) is a 1 (land), so ans  $+= 4 \Rightarrow$  ans = 4.

• The second cell (0,1) is a 0 (water), so no change to ans.

• The third cell (0,2) is a 1 (land), so ans  $+= 4 \Rightarrow$  ans = 8.

• The third cell (1,2) is a 0 (water), so no change to ans.

• The third cell (2,2) is a 0 (water), so no change to ans.

Thus, the perimeter of the island in the given grid is 12.

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

# Get the number of rows and columns of the grid

# If we encounter a land cell

perimeter -= 2

// Function to calculate the perimeter of the island.

public int islandPerimeter(int[][] grid) {

// Get the number of rows in the grid.

// Get the number of columns in the grid.

for (int j = 0; j < cols; j++) {

if (grid[i][i] == 1) {

perimeter += 4;

// Iterate through the grid using nested loops.

perimeter -= 2;

perimeter -= 2;

// Return the total perimeter of the island.

for (let col = 0: col < width: ++col) {</pre>

topNeighbor = grid[row - 1][col];

leftNeighbor = grid[row][col - 1];

if (currentCell !== topNeighbor) ++perimeter;

// Account for the last row and last column edges

if (currentCell !== leftNeighbor) ++perimeter;

return perimeter; // Return the total perimeter of the islands

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

# Get the number of rows and columns of the grid

# If we encounter a land cell

perimeter −= 2

# Return the total perimeter of the island

# Add 4 sides to the perimeter

if grid[row][col] == 1:

perimeter += 4

rows, cols = len(grid), len(grid[0])

# Go through each cell in the grid

for col in range(cols):

# Initialize perimeter count

for row in range(rows):

perimeter = 0

if (row > 0) {

**if** (col > 0) {

for (let i = 0; i < height; ++i) {</pre>

for (let j = 0; j < width; ++j) {

class Solution:

let topNeighbor = 0, // Variable to track the top neighbor's value

// Check if the top neighbor exists, and if so, get its value

// Check if the left neighbor exists, and if so, get its value

let currentCell = grid[row][col]; // Current cell value

leftNeighbor = 0; // Variable to track the left neighbor's value

// Compare current cell with the top and left cells; increment perimeter accordingly

if (grid[i][width - 1] === 1) ++perimeter; // Increment if last column cell is land

if (grid[height - 1][j] === 1) ++perimeter; // Increment if last row cell is land

// Initialize perimeter sum to 0.

for (int i = 0; i < rows; i++) {

int perimeter = 0;

return perimeter;

int rows = grid.length;

int cols = grid[0].length;

# Add 4 sides to the perimeter

if grid[row][col] == 1:

perimeter += 4

rows, cols = len(grid), len(grid[0])

# Initialize perimeter count

Solution Implementation

perimeter = 0

**Python** 

class Solution:

with adjacent land. Subtract Shared Edges: Subtractions are accounted for when checking for adjacent land.

Check for Adjacent Land: We have also done this part during our iteration and subtracted 2 from ans for every shared edge

# Go through each cell in the grid for row in range(rows): for col in range(cols):

# subtract 2 from the perimeter (common side with the bottom cell)

**Return Perimeter Count:** After processing the entire grid, the total perimeter ans is 12.

# subtract 2 from the perimeter (common side with the right cell) if col < cols - 1 and grid[row][col + 1] == 1:</pre> perimeter -= 2 # Return the total perimeter of the island return perimeter

# If there is a land cell to the right of the current one.

# If there is a land cell below the current one.

// Check if the current cell is land (1 indicates land).

if (i < rows - 1 && grid[i + 1][j] == 1) {</pre>

if (i < cols - 1 && grid[i][j + 1] == 1) {</pre>

// Add 4 for each land cell as it could potentially contribute 4 sides to the perimeter.

// If there is land directly to the right of the current land, subtract 2 from perimeter count

// If there is land directly below the current land, subtract 2 from perimeter count

// (one for the current cell's bottom side and one for the bottom cell's top side).

// (one for the current cell's right side and one for the right cell's left side).

if row < rows - 1 and grid[row + 1][col] == 1:</pre>

Java

class Solution {

```
C++
class Solution {
public:
    // Function to calculate the perimeter of islands in a grid.
    int islandPerimeter(vector<vector<int>>& grid) {
        // m is the number of rows in the grid.
        int rowCount = grid.size();
        // n is the number of columns in the grid.
        int columnCount = grid[0].size();
        // Initialize the perimeter result to 0.
        int perimeter = 0;
        // Iterate over each cell in the grid.
        for (int row = 0; row < rowCount; ++row) {</pre>
            for (int column = 0; column < columnCount; ++column) {</pre>
                // Check if the current cell is part of an island.
                if (grid[row][column] == 1) {
                    // Each island cell contributes 4 to the perimeter.
                    perimeter += 4;
                    // If the cell below the current one is also part of the island,
                    // reduce the perimeter by 2 (since two sides are internal and do not contribute to the perimeter).
                    if (row < rowCount - 1 && grid[row + 1][column] == 1) perimeter -= 2;</pre>
                    // If the cell to the right of the current one is also part of the island,
                    // reduce the perimeter by 2 for the same reason.
                    if (column < columnCount - 1 && grid[row][column + 1] == 1) perimeter -= 2;</pre>
        // Return the total perimeter calculated.
        return perimeter;
};
TypeScript
// Function to calculate the perimeter of islands.
// The grid is represented by a 2D array where 1 indicates land and 0 indicates water.
function islandPerimeter(grid: number[][]): number {
    let height = grid.length, // The height of the grid
        width = grid[0].length; // The width of the grid
    let perimeter = 0; // Initialize perimeter counter
    // Iterate over each cell in the grid
    for (let row = 0; row < height; ++row) {</pre>
```

```
# subtract 2 from the perimeter (common side with the bottom cell)
if row < rows - 1 and grid[row + 1][col] == 1:</pre>
    perimeter -= 2
# If there is a land cell to the right of the current one.
# subtract 2 from the perimeter (common side with the right cell)
```

return perimeter

**Time and Space Complexity** 

The time complexity of the given code is 0(m \* n) where m is the number of rows and n is the number of columns in the grid. This is because there is a nested loop which iterates over each cell in the grid exactly once.

The space complexity of the code is 0(1) since it only uses a constant amount of additional space. The variable ans is updated in place and no additional space that scales with the size of the input is allocated.

# If there is a land cell below the current one.

if col < cols - 1 and grid[row][col + 1] == 1:</pre>