VMES Grinds

# Leetcode: 463
# Island Perimeter

*Noel*

# 463. Island Perimeter

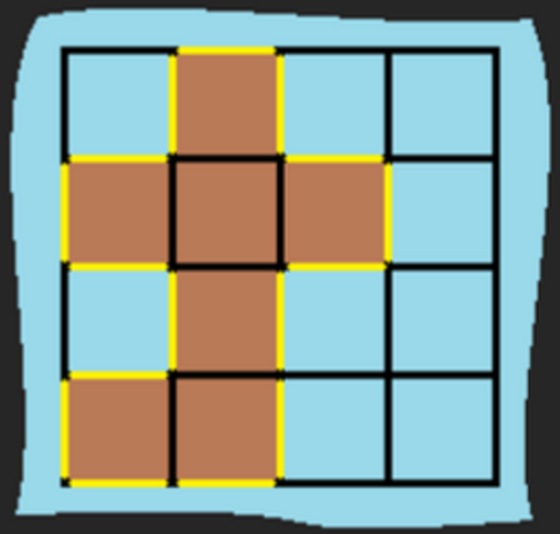
Easy | Topics | Companies

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`.

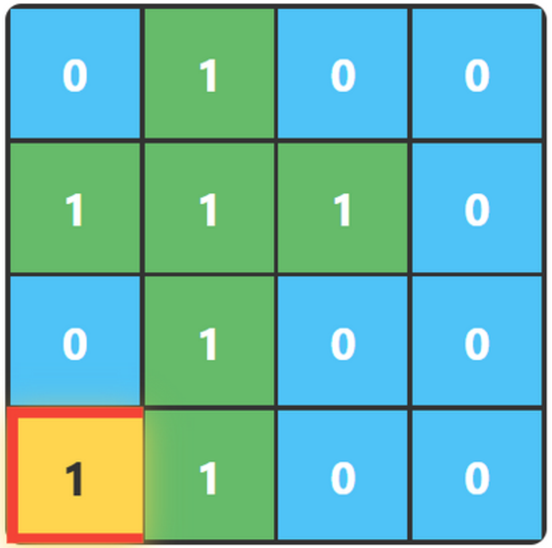
# Explanation

# 🏝️ Island Perimeter Visualization

Water (0) Land (1) Current Cell Perimeter Edge

⬅ Previous Next ➡ 🔄 Reset

Step 6 of 7



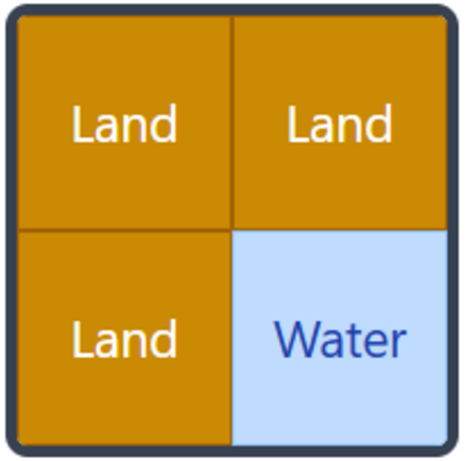
# Initial Approach

VMES Grinds

```python
class Solution:
    def islandPerimeter(self, grid: List[List[int]]) -> int:
            rows, cols = len(grid), len(grid[0])
            perimeter = 0
            land_count = 0

            total_land = sum(sum(row) for row in grid)

            for i in range(rows):
                for j in range(cols):
                    if grid[i][j] == 1:
                        for di, dj in [(-1,0), (1,0), (0,-1), (0,1)]:
                            ni, nj = i + di, j + dj
                            if ni < 0 or ni >= rows or nj < 0 or nj >= cols or grid[ni][nj] == 0:
                                perimeter += 1

                        land_count += 1
                        if land_count == total_land:
                            return perimeter

            return perimeter
```

# Optimized Approach

# The "+4, -2" Logic




**Perimeter:** 0

Press 'Next' to start.

Previous | Next | Reset

```python
class Solution:
    def islandPerimeter(self, grid: List[List[int]]) -> int:
        perimeter = 0
        rows, cols = len(grid), len(grid[0])

        for r in range(rows):
            for c in range(cols):
                if grid[r][c] == 1:
                    perimeter += 4
                    if r > 0 and grid[r-1][c] == 1:
                        perimeter -= 2
                    if c > 0 and grid[r][c-1] == 1:
                        perimeter -= 2

        return perimeter
```

VMES Grinds

# Time and Space Complexity Analysis

```python
class Solution:
    def islandPerimeter(self, grid: List[List[int]]) -> int:
        rows, cols = len(grid), len(grid[0])
        perimeter = 0
        land_count = 0

        total_land = sum(sum(row) for row in grid)

        for i in range(rows):
            for j in range(cols):
                if grid[i][j] == 1:
                    for di, dj in [(-1,0), (1,0), (0,-1), (0,1)]:
                        ni, nj = i + di, j + dj
                        if ni < 0 or ni >= rows or nj < 0 or nj >= cols or grid[ni][nj] == 0:
                            perimeter += 1

                    land_count += 1
                    if land_count == total_land:
                        return perimeter

        return perimeter
```

Space - O(1)

Time - O(rows x columns) + O(rows x columns)

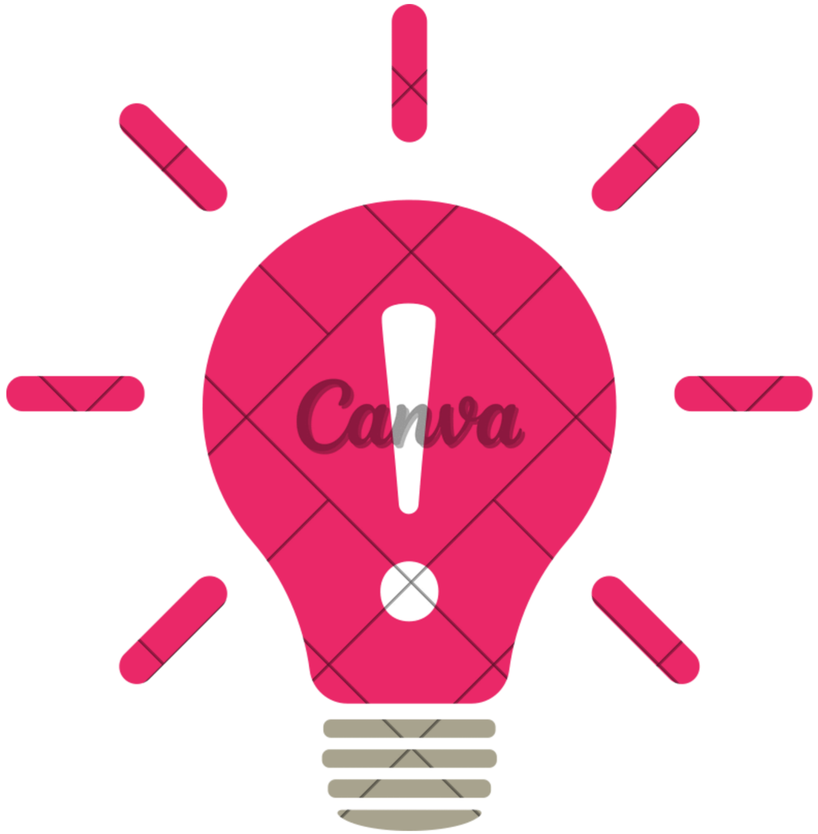
```python
class Solution:
    def islandPerimeter(self, grid: List[List[int]]) -> int:
        perimeter = 0
        rows, cols = len(grid), len(grid[0])

        for r in range(rows):
            for c in range(cols):
                if grid[r][c] == 1:
                    perimeter += 4
                    if r > 0 and grid[r-1][c] == 1:
                        perimeter -= 2
                    if c > 0 and grid[r][c-1] == 1:
                        perimeter -= 2

        return perimeter
```

Space - O(1)

Time - O(rows x columns)

# Key Takeaway

VMES Grinds




Single Pass vs. Multiple Passes

The first solution is faster as it finds the perimeter in one pass, while the second needs two (one to count land, one to check neighbors).

Fewer Operations

The first solution is more efficient by performing at most 2 checks (up, left) per land cell, whereas the second always performs 4.