Articles › 463. Island Perimeter

Previous (/articles/4sum-ii/) Next (/articles/sum-of-two-integers/)

463. Island Perimeter (/problems/island-perimeter/)

June 7, 2020 | 7.6K views

Average Rating: 4.90 (19 votes)

You are given a map in form of a two-dimensional integer grid where 1 represents land and 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).

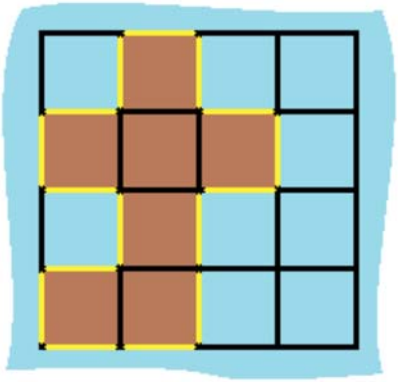
The island doesn't have "lakes" (water inside that isn't connected to the water around the island). One cell is a square with side length 1. The grid is rectangular, width and height don't exceed 100. Determine the perimeter of the island.

Example:

```
Input:
[[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 below:
```



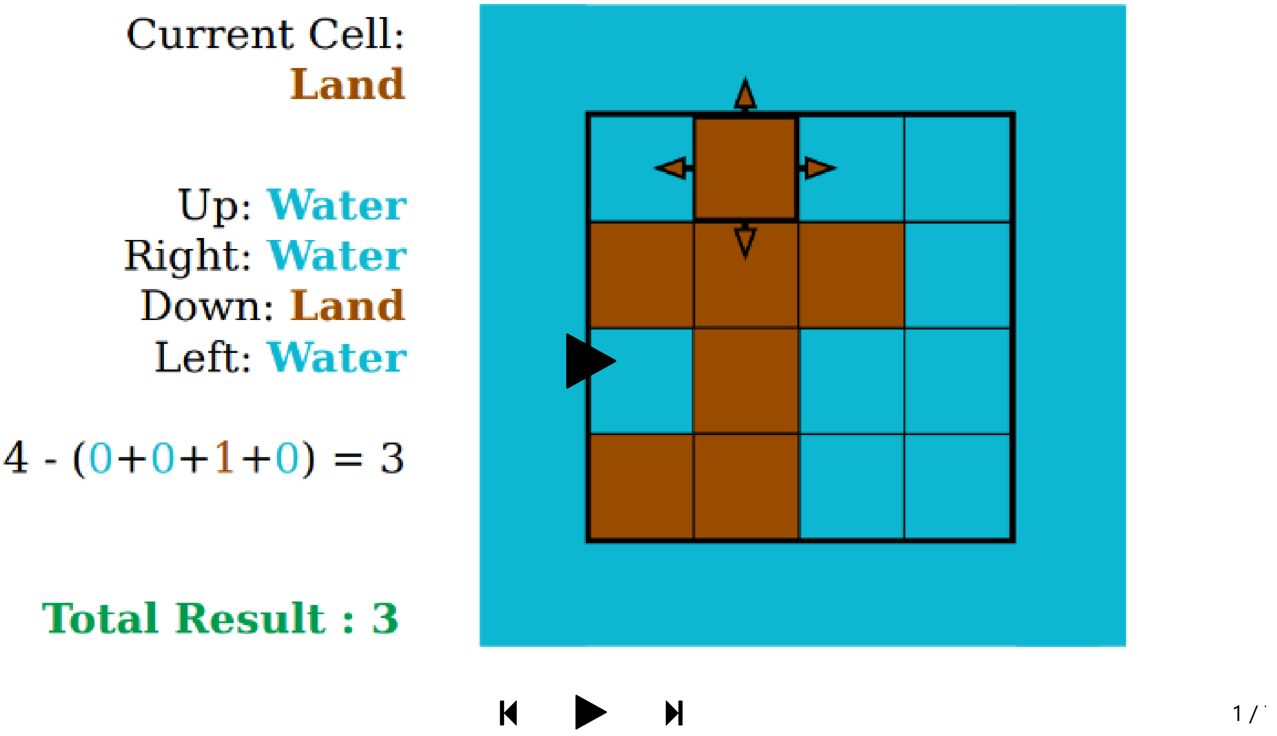
Solution

Approach 1: Simple Counting

Intuition

Go through every cell on the grid and whenever you are at cell `1` (land cell), look for surrounding (`UP` , `RIGHT` , `DOWN` , `LEFT`) cells. A land cell without any surrounding land cell will have a perimeter of `4` . Subtract `1` for each surrounding land cell.

When you are at cell `0` (water cell), you don't need to do anything. Just proceed to another cell.



Current Cell: Land

Up: Water
Right: Water
Down: Land
Left: Water

4 - (0+0+1+0) = 3

Total Result : 3

Implementation

Java Python3 Copy

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

        int rows = grid.length;
        int cols = grid[0].length;

        int up, down, left, right;
        int result = 0;

        for (int r = 0; r < rows; r++) {
            for (int c = 0; c < cols; c++) {
                if (grid[r][c] == 1) {
                    if (r == 0) { up = 0; }
                    else { up = grid[r-1][c]; }

                    if (c == 0) { left = 0; }
                    else { left = grid[r][c-1]; }

                    if (r == rows-1) { down = 0; }
                    else { down = grid[r+1][c]; }

                    if (c == cols-1) { right = 0; }
                    else { right = grid[r][c+1]; }

                    result += 4-(up+left+right+down);
                }
            }
        }

        return result;
    }
}
```

Complexity Analysis

- Time complexity : $O(mn)$ where m is the number of rows of the grid and n is the number of columns of the grid. Since two `for` loops go through all the cells on the grid, for a two-dimensional grid of size $m \times n$, the algorithm would have to check mn cells.

- Space complexity : $O(1)$. Only the `result` variable is updated and there is no other space requirement.

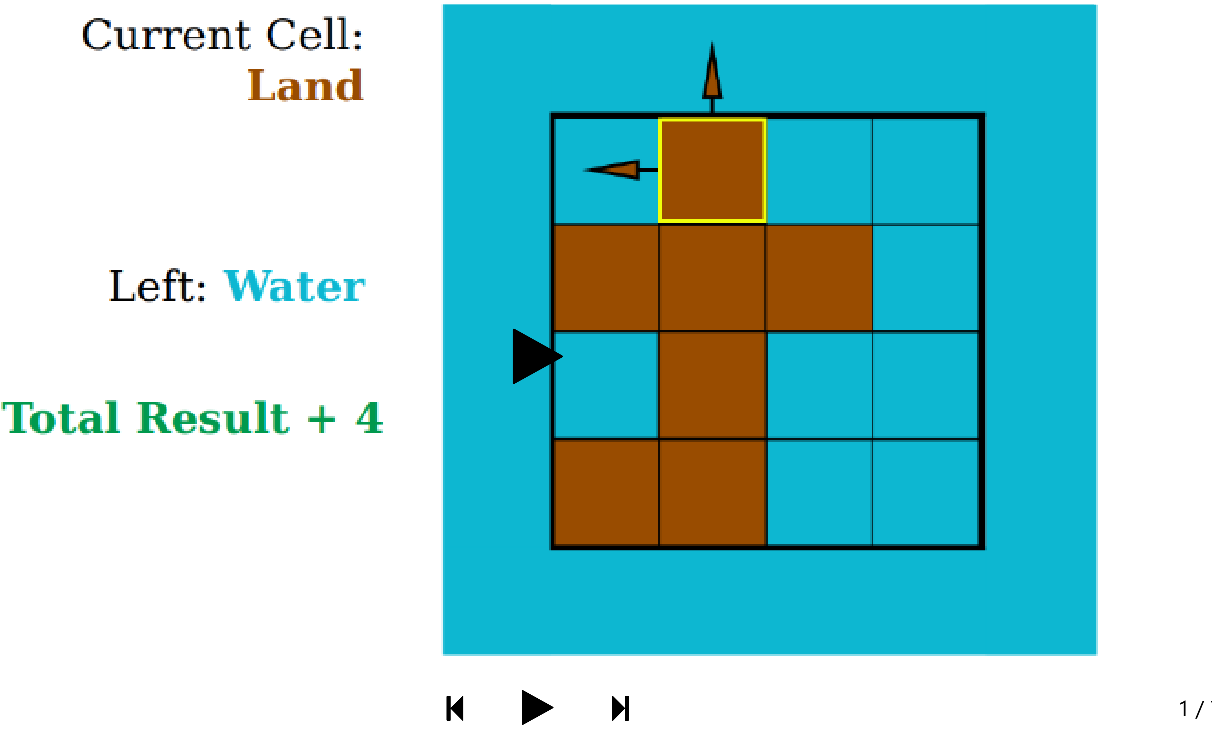
Approach 2: Better Counting

Approach 2 has the same time and space complexity as Approach 1. Even though they have the same time and space complexities, Approach 2 is slightly more efficient than the Approach 1. Rather than checking 4 surrounding neighbors, we only need to check two neighbors (`LEFT` and `UP`) in Approach 2.

Intuition

Since we are traversing the grid from left to right, and from top to bottom, for each land cell we are currently at, we only need to check whether the `LEFT` and `UP` cells are land cells with a slight modification on previous approach.

- As you go through each cell on the grid, treat all the land cells as having a perimeter of `4` and add that up to the accumulated `result`.
- If that land cell has a neighboring land cell, remove `2` sides (one from each land cell) which will be touching between these two cells.
  - If your current land cell has a `UP` land cell, subtract `2` from your accumulated `result`.
  - If your current land cell has a `LEFT` land cell, subtract `2` from your accumulated `result`.



Implementation

Java Python3 Copy

```java
class Solution {
    public int islandPerimeter(int[][] grid) {
        int rows = grid.length;
        int cols = grid[0].length;

        int result = 0;
        for (int r = 0; r < rows; r++) {
            for (int c = 0; c < cols; c++) {
                if (grid[r][c] == 1) {
                    result += 4;

                    if (r > 0 && grid[r-1][c] == 1) {
                        result -= 2;
                    }

                    if (c > 0 && grid[r][c-1] == 1) {
                        result -= 2;
                    }
                }
            }
        }

        return result;
    }
}
```

Complexity Analysis

- Time complexity : $O(mn)$ where m is the number of rows of the grid and n is the number of columns of the grid. Since two `for` loops go through all the cells on the grid, for a two-dimensional grid of size $m \times n$, the algorithm would have to check mn cells.
- Space complexity : $O(1)$. Only the `result` variable is updated and there is no other space requirement.

Rate this article:

Previous (/articles/4sum-ii/)

Next (/articles/sum-of-two-integers/)

Comments: 3

Sort By

Type comment here... (Markdown is supported)

Preview

Post

(/thepatriot)

thepatriot (/thepatriot) ★ 250 4 hours ago

why is this question labeled with the "Hash Table" tag?

1 Share Reply

SHOW 2 REPLIES

(/praveenpurwar2004)

praveenpurwar2004 (/praveenpurwar2004) ★ 5 June 24, 2020 1:53 AM

Easy Python solution using direction logic :

```
d=[(1,0),(-1,0),(0,-1),(0,1)]
        # 0,0 0,1 0,2
        # 1,0 1,1 1,2
```

Read More

0 Share Reply

SHOW 1 REPLY

(/_noexcuses)

_noexcuses (/_noexcuses) ★ 235 an hour ago

Concise C++ using counting
Complexity: linear in the total size (rows * columns) of grid

```
  int islandPerimeter(vector<vector<int>>& grid) {
    int per = 0, m = grid.size(), n = grid[0].size();
```

Read More

0 Share Reply

Help Center (/support/) | Terms (/terms/) | Privacy (/privacy/) United States (/region/)