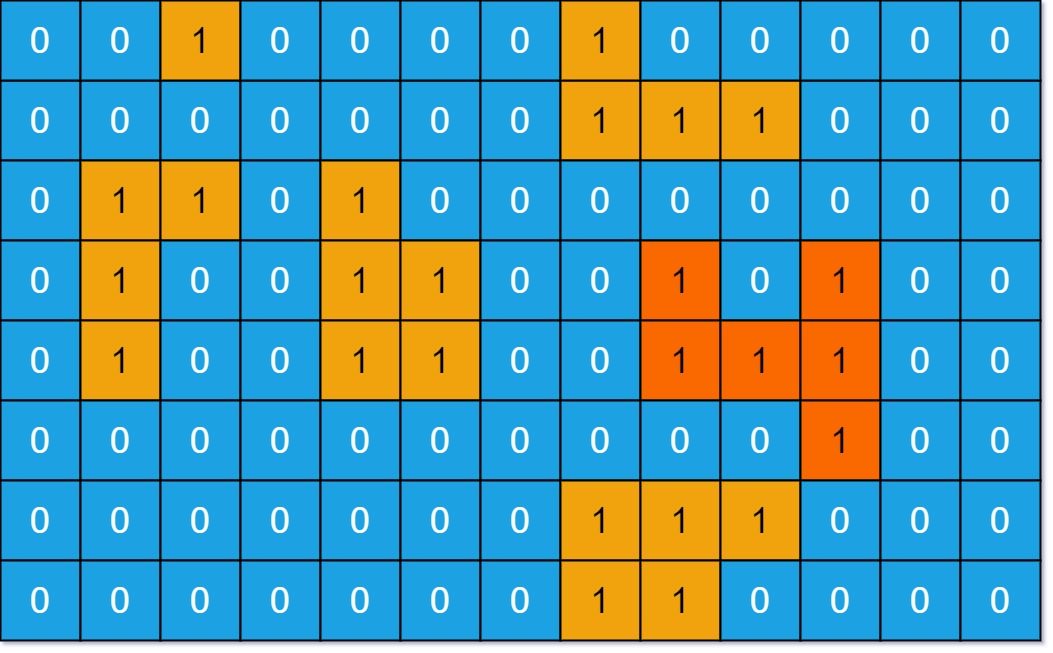
<https://leetcode.com/problems/max-area-of-island/description/>

You are given an m x n binary matrix grid. An island is a group of 1's (representing land) connected **4-directionally** (horizontal or vertical.) You may assume all four edges of the grid are surrounded by water.

The **area** of an island is the number of cells with a value 1 in the island.

Return the maximum **area** of an island in grid. If there is no island, return 0.

**Example 1:**



**Input:** grid = [[0,0,1,0,0,0,0,1,0,0,0,0,0],[0,0,0,0,0,0,0,1,1,1,0,0,0],[0,1,1,0,1,0,0,0,0,0,0,0,0],[0,1,0,0,1,1,0,0,1,0,1,0,0],[0,1,0,0,1,1,0,0,1,1,1,0,0],[0,0,0,0,0,0,0,0,0,0,1,0,0],[0,0,0,0,0,0,0,1,1,1,0,0,0],[0,0,0,0,0,0,0,1,1,0,0,0,0]]

**Output:** 6

**Explanation:** The answer is not 11, because the island must be connected 4-directionally.

**Example 2:**

**Input:** grid = [[0,0,0,0,0,0,0,0]]

**Output:** 0

**Constraints:**

m == grid.length

n == grid[i].length

1 <= m, n <= 50

grid[i][j] is either 0 or 1.

**Attempt 1: 2024-12-21**

**Solution 1: DFS (10 min)**

**特别注意返回 int 值的 DFS 的写法，变量 area 设置的位置，如何配合后续的更深层的 DFS，**

class Solution {

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

        int m = grid.length;

        int n = grid[0].length;

        int maxArea = 0;

        for(int i = 0; i < m; i++) {

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

                if(grid[i][j] == 1) {

                    maxArea = Math.max(maxArea, helper(grid, i, j, m, n));

                }

            }

        }

        return maxArea;

    }

    int[] dx = new int[] {0, 0, 1, -1};

    int[] dy = new int[] {1, -1, 0, 0};

    private int helper(int[][] grid, int x, int y, int m, int n) {

        if(x < 0 || x >= m || y < 0 || y >= n || grid[x][y] == 0) {

            return 0;

        }

        // Mark the cell as visited by setting it to 0

        grid[x][y] = 0;

        // Area of the current cell is 1 plus the area from all 4 directions

        int area = 1;

        for(int k = 0; k < 4; k++) {

            int new\_x = x + dx[k];

            int new\_y = y + dy[k];

            area += helper(grid, new\_x, new\_y, m, n);

        }

        return area;

    }

}

Time Complexity: O(m \* n), every cell in the grid is visited at most once.

Space Complexity: O(m \* n)

**Solution 2: BFS (10 min)**

class Solution {

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

        int[] dx = {0, 0, 1, -1};

        int[] dy = {1, -1, 0, 0};

        int m = grid.length;

        int n = grid[0].length;

        int maxArea = 0;

        for(int i = 0; i < m; i++) {

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

                if(grid[i][j] == 1) {

                    int area = 0;

                    Queue<int[]> queue = new LinkedList<>();

                    queue.offer(new int[]{i, j});

                    // Mark as visited

                    grid[i][j] = 0;

                    while(!queue.isEmpty()) {

                        int[] cur = queue.poll();

                        area++;

                        for(int k = 0; k < 4; k++) {

                            int new\_x = cur[0] + dx[k];

                            int new\_y = cur[1] + dy[k];

                            // Check boundaries and if the cell is land

                                queue.offer(new int[]{new\_x, new\_y});

                                // Mark as visited

                                grid[new\_x][new\_y] = 0;

                            }

                        }

                    }

                    maxArea = Math.max(maxArea, area);

                }

            }

        }

        return maxArea;

    }

}

Time Complexity: O(m \* n), every cell in the grid is visited at most once.

Space Complexity: O(m \* n)

**Refer to chatGPT**

**DFS Solution**

class Solution {

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

int maxArea = 0;

int rows = grid.length;

int cols = grid[0].length;

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

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

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

maxArea = Math.max(maxArea, dfs(grid, i, j));

}

}

}

return maxArea;

}

private int dfs(int[][] grid, int row, int col) {

// Check boundaries and if the cell is water or already visited

if (row < 0 || row >= grid.length || col < 0 || col >= grid[0].length || grid[row][col] == 0) {

return 0;

}

// Mark the cell as visited by setting it to 0

grid[row][col] = 0;

// Area of the current cell is 1 plus the area from all 4 directions

int area = 1;

area += dfs(grid, row + 1, col); // Down

area += dfs(grid, row - 1, col); // Up

area += dfs(grid, row, col + 1); // Right

area += dfs(grid, row, col - 1); // Left

return area;

}

}

**BFS Solution**

class Solution {

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

int maxArea = 0;

int rows = grid.length;

int cols = grid[0].length;

// Directions for moving up, down, left, and right

int[] dx = {0, 0, 1, -1};

int[] dy = {1, -1, 0, 0};

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

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

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

int area = 0;

Queue<int[]> queue = new LinkedList<>();

queue.offer(new int[]{i, j});

grid[i][j] = 0; // Mark as visited

while (!queue.isEmpty()) {

int[] cell = queue.poll();

area++;

for (int k = 0; k < 4; k++) {

int newRow = cell[0] + dx[k];

int newCol = cell[1] + dy[k];

// Check boundaries and if the cell is land

queue.offer(new int[]{newRow, newCol});

grid[newRow][newCol] = 0; // Mark as visited

}

}

}

maxArea = Math.max(maxArea, area);

}

}

}

return maxArea;

}

}

**Explanation of the Code**

Traversal:

Both solutions iterate through each cell in the grid. If the cell contains 1 (land), it calculates the area of the connected island.

DFS/BFS:

DFS uses recursion to explore all connected cells and compute the area.

BFS uses a queue to iteratively explore neighboring cells.

Marking Visited:

Instead of using a separate visited array, the grid itself is updated by marking visited land cells as 0.

Maximum Area:

For each island, the computed area is compared with the current maxArea to keep track of the largest island.

**Refer to**

[L200.Number of Islands (Ref.L1568)](note://33605BDB820F44709185B3524EB4775A)

[L1568.Minimum Number of Days to Disconnect Island (Ref.L200)](note://WEBe1f9dea2f2571376df84b64404041136)