Max Area of Island

Given a non-empty 2D array `grid` of 0's and 1's, 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.

Find the maximum area of an island in the given 2D array. (If there is no island, the maximum area is 0.)

Example 1:

```
[[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]]
```

Given the above grid, return `6`. Note the answer is not 11, because the island must be connected 4-directionally.

Example 2:

```
[[0,0,0,0,0,0,0,0]]
```

Given the above grid, return `0`.

Note: The length of each dimension in the given `grid` does not exceed 50.

Solution 1

The idea is to count the area of each island using dfs. During the dfs, we set the value of each point in the island to `0`. The time complexity is `O(mn)`.

Java version:

```
    public int maxAreaOfIsland(int[][] grid) {
        int max_area = 0;
        for(int i = 0; i < grid.length; i++)
            for(int j = 0; j < grid[0].length; j++)
                if(grid[i][j] == 1)max_area = Math.max(max_area, AreaOfIsland(grid,
i, j));
        return max_area;
    }

    public int AreaOfIsland(int[][] grid, int i, int j){
        if( i >= 0 && i < grid.length && j >= 0 && j < grid[0].length && grid[i][j]
== 1){
            grid[i][j] = 0;
            return 1 + AreaOfIsland(grid, i+1, j) + AreaOfIsland(grid, i-1, j) + Are
aOfIsland(grid, i, j-1) + AreaOfIsland(grid, i, j+1);
        }
        return 0;
    }
```

C++ version:

```
    int maxAreaOfIsland(vector<vector<int>>& grid) {
        int max_area = 0;
        for(int i = 0; i < grid.size(); i++)
            for(int j = 0; j < grid[0].size(); j++)
                if(grid[i][j] == 1)max_area = max(max_area, AreaOfIsland(grid, i, j)
);
        return max_area;
    }

    int AreaOfIsland(vector<vector<int>>& grid, int i, int j){
        if( i >= 0 && i < grid.size() && j >= 0 && j < grid[0].size() && grid[i][j]
== 1){
            grid[i][j] = 0;
            return 1 + AreaOfIsland(grid, i+1, j) + AreaOfIsland(grid, i-1, j) + Are
aOfIsland(grid, i, j-1) + AreaOfIsland(grid, i, j+1);
        }
        return 0;
    }
```

written by Vincent Cai original link here

Solution 2

```
    public int maxAreaOfIsland(int[][] grid) {
        if (grid == null || grid.length == 0) {
            return 0;
        }
        int m = grid.length;
        int n = grid[0].length;
        int max = 0;
        for (int i = 0; i < m; i++) {
            for (int j = 0; j < n; j++) {
                if (grid[i][j] == 1) {
                    int area = dfs(grid, i, j, m, n, 0);
                    max = Math.max(area, max);
                }
            }
        }
        return max;
    }

    int dfs(int[][] grid, int i, int j, int m, int n, int area) {
        if (i < 0 || i >= m || j < 0 || j >= n || grid[i][j] == 0) {
            return area;
        }
        grid[i][j] = 0;
        area++;
        area = dfs(grid, i + 1, j, m, n, area);
        area = dfs(grid, i, j + 1, m, n, area);
        area = dfs(grid, i - 1, j, m, n, area);
        area = dfs(grid, i, j - 1, m, n, area);
        return area;
    }
```

written by fishercoder original link here

Solution 3

```
def maxAreaOfIsland(self, grid):
    grid = {i + j*1j: val for i, row in enumerate(grid) for j, val in enumerate(row)
}
    def area(z):
        return grid.pop(z, 0) and 1 + sum(area(z + 1j**k) for k in range(4))
    return max(map(area, set(grid)))
```

Complex numbers making things simple...

written by StefanPochmann original link here