Number of Distinct Islands

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.

Count the number of **distinct** islands. An island is considered to be the same as another if and only if one island can be translated (and not rotated or reflected) to equal the other.

Example 1:

```
11000
11000
00011
00011
```

Given the above grid map, return `1`.

Example 2:

```
11011
10000
00001
11011
```

Given the above grid map, return `3`.

Notice that:

```
11
1
```

and

```
 1
11
```

are considered different island shapes, because we do not consider reflection / rotation.

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

Solution 1

Java

```
class Solution {
    private static int[][] delta = { {0, 1}, {1, 0}, {0, -1}, {-1, 0} };

    public int numDistinctIslands(int[][] grid) {
        int m = grid.length, n = grid[0].length;
        Set<List<List<Integer>>> islands = new HashSet<>();
        for (int i = 0; i < m; i++) {
            for (int j = 0; j < n; j++) {
                List<List<Integer>> island = new ArrayList<>();
                if (dfs(i, j, i, j, grid, m, n, island))
                    islands.add(island);
            }
        }
        return islands.size();
    }

    private boolean dfs(int i0, int j0, int i, int j, int[][] grid, int m, int n, Li
st<List<Integer>> island) {
        if (i < 0 || m <= i || j < 0 || n <= j || grid[i][j] <= 0) return false;
        island.add(Arrays.asList(i - i0, j - j0));
        grid[i][j] *= -1;
        for (int d = 0; d < 4; d++) {
            dfs(i0, j0, i + delta[d][0], j + delta[d][1], grid, m, n, island);
        }
        return true;
    }
}
```

C++

```
class Solution {
public:
    int numDistinctIslands(vector<vector<int>>& grid) {
        int m = grid.size(), n = grid[0].size();
        set<vector<vector<int>>> islands;
        for (int i = 0; i < m; i++) {
            for (int j = 0; j < n; j++) {
                vector<vector<int>> island;
                if (dfs(i, j, i, j, grid, m, n, island))
                    islands.insert(island);
            }
        }
        return islands.size();
    }

private:
    int delta[4][2] = { 0, 1, 1, 0, 0, -1, -1, 0};

    bool dfs(int i0, int j0, int i, int j, vector<vector<int>>& grid, int m, int n,
vector<vector<int>>& island) {
        if (i < 0 || m <= i || j < 0 || n <= j || grid[i][j] <= 0) return false;
        island.push_back({i - i0, j - j0});
        grid[i][j] *= -1;
        for (int d = 0; d < 4; d++) {
            dfs(i0, j0, i + delta[d][0], j + delta[d][1], grid, m, n, island);
        }
        return true;
    }
};
```

written by alexander original link here

Solution 2

```
class Solution {

    int[][] dirs= new int[][]{{1,0},{0,1},{-1,0},{0,-1}};
    public int numDistinctIslands(int[][] grid) {
         Set<String> set= new HashSet<>();
        int res=0;

        for(int i=0;i<grid.length;i++){
            for(int j=0;j<grid[0].length;j++){
                if(grid[i][j]==1) {
                    StringBuilder sb= new StringBuilder();
                    helper(grid,i,j,0,0, sb);
                    String s=sb.toString();
                    if(!set.contains(s)){
                    res++;
                    set.add(s);
}
                }
            }
        }
            return res;
    }

    public  void helper(int[][] grid,int i,int j, int xpos, int ypos,StringBuilder s
b){
        grid[i][j]=0;
        sb.append(xpos+""+ypos);
        for(int[] dir : dirs){
            int x=i+dir[0];
            int y=j+dir[1];
            if(x<0 || y<0 || x>=grid.length || y>=grid[0].length || grid[x][y]==0) c
ontinue;
            helper(grid,x,y,xpos+dir[0],ypos+dir[1],sb);
        }
    }
}
```

written by kay_deep original link here

Solution 3

This question is very similar to the Max Area of Island question but here instead of counting the area for each island, we find out the shape of each island.

The shape of the island can be represented by taking the relative position of the connected cells from the leftmost cell on the top row of the island (the first cell of each island we will visit). For each island we visit, we are guaranteed to visit the top row's leftmost cell if we iterate the matrix row by row, left to right direction and we will get the same order of cells for islands of the same shape if we perform the search in a consistent manner.

Here are some examples of how to represent the shape of each island by using cell positions relative to the top left cell.

```
# First coordinate is row difference,
# second coordinate is column difference.
11 -> ((0, 1)) # One cell to the right

11 -> ((0, 1), (1, 1)) # One cell to the right, one cell to the right and bottom
01
```

```
class Solution(object):
    def numDistinctIslands(self, grid):
        """
        :type grid: List[List[int]]
        :rtype: int
        """
        island_shapes = set()
        rows, cols = len(grid), len(grid[0])
        def dfs(i, j, positions, rel_pos):
            grid[i][j] = -1
            for direction in ((0, 1), (1, 0), (-1, 0), (0, -1)):
                next_i, next_j = i + direction[0], j + direction[1]
                if (0 <= next_i < rows and 0 <= next_j < cols) and grid[next_i][nex
t_j] == 1:
                    new_rel_pos = (rel_pos[0] + direction[0], rel_pos[1] + directio
n[1])
                    positions.append(new_rel_pos)
                    dfs(next_i, next_j, positions, new_rel_pos)
        for i in range(rows):
            for j in range(cols):
                if grid[i][j] == 1:
                    positions = []
                    dfs(i, j, positions, (0, 0))
                    island_shapes.add(tuple(positions))
        return len(island_shapes)
```

written by yangshun original link here