# 694. Number of Distinct Islands

Oct. 7, 2017 | 34.9K views

Average Rating: 3.87 (30 votes)

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.

## Approach #1: Hash By Local Coordinates [Accepted]

**Intuition and Algorithm**

At the beginning, we need to find every island, which we can do using a straightforward depth-first search. The hard part is deciding whether two islands are the same.

Since two islands are the same if one can be translated to match another, let's translate every island so the top-left corner is `(0, 0)`. For example, if an island is made from squares `[(2, 3), (2, 4), (3, 4)]`, we can think of this shape as `[(0, 0), (0, 1), (1, 1)]` when anchored at the top-left corner.

From there, we only need to check how many unique shapes there ignoring permutations (so `[(0, 0), (0, 1)]` is the same as `[(0, 1), (1, 0)]`). We use sets directly as we have showcased below, but we could have also sorted each list and put those sorted lists in our set structure `shapes`.

In the Java solution, we converted our tuples `(r - r0, c - c0)` to integers. We multiplied the number of rows by `2 * grid[0].length` instead of `grid[0].length` because our local row-coordinate could be negative.

Java Python Copy

```
class Solution(object):
    def numDistinctIslands(self, grid):
        seen = set()
        def explore(r, c, r0, c0):
            if (0 <= r < len(grid) and 0 <= c < len(grid[0]) and
                    grid[r][c] and (r, c) not in seen):
                seen.add((r, c))
                shape.add((r - r0, c - c0))
                explore(r+1, c, r0, c0)
                explore(r-1, c, r0, c0)
                explore(r, c+1, r0, c0)
                explore(r, c-1, r0, c0)

        shapes = set()
        for r in range(len(grid)):
            for c in range(len(grid[0])):
                shape = set()
                explore(r, c, r, c)
                if shape:
                    shapes.add(frozenset(shape))
        return len(shapes)
```

**Complexity Analysis**

- Time Complexity: $O(R * C)$, where $R$ is the number of rows in the given `grid`, and $C$ is the number of columns. We visit every square once.
- Space complexity: $O(R * C)$, the space used by `seen` to keep track of visited squares, and `shapes`.

## Approach #2: Hash By Path Signature [Accepted]

**Intuition and Algorithm**

When we start a depth-first search on the top-left square of some island, the path taken by our depth-first search will be the same if and only if the shape is the same. We can exploit this by recording the path we take as our shape - keeping in mind to record both when we enter and when we exit the function. The rest of the code remains as in *Approach #1*.

Java Python Copy

```
class Solution(object):
    def numDistinctIslands(self, grid):
        seen = set()
        def explore(r, c, di = 0):
            if (0 <= r < len(grid) and 0 <= c < len(grid[0]) and
                    grid[r][c] and (r, c) not in seen):
                seen.add((r, c))
                shape.append(di)
                explore(r+1, c, 1)
                explore(r-1, c, 2)
                explore(r, c+1, 3)
                explore(r, c-1, 4)
                shape.append(0)

        shapes = set()
        for r in xrange(len(grid)):
            for c in xrange(len(grid[0])):
                shape = []
                explore(r, c)
                if shape:
                    shapes.add(tuple(shape))

        return len(shapes)
```

**Complexity Analysis**

- Time and Space Complexity: $O(R * C)$. The analysis is the same as in *Approach #1*.

Analysis written by: @awice

Rate this article:

Previous Next

## Comments: 29

Sort By

Type comment here... (Markdown is supported)

Preview Post

**XinyuHou** ★21 May 10, 2018 12:43 AM

Firstly, thanks for the answer.

> From there, we only need to check how many unique shapes there ignoring permutations (so [[0, 0], [0, 1]] is the same as [[0, 1], [1, 0]]).

Read More

21 Share Reply

SHOW 5 REPLIES

**MockCode** ★26 October 8, 2017 9:29 PM Report

Why do you have "shape.add(0)" in the function explore of the second approach?

20 Share Reply

SHOW 2 REPLIES

**jwgoh** ★16 October 8, 2017 12:47 PM

> For example, if an island is made from squares [(2, 3), (2, 4), (3, 4)], we can think of this shape as [(0, 0), (0, 1), (1, 0)] when anchored at the top-left corner.

Should be [(0, 0), (0, 1), **(1, 1)**]

9 Share Reply

**awice** ★4246 October 9, 2017 12:23 AM

@jwgoh Corrected. thanks.

@kk636 ( `shape.add(0)` )

Say we have shapes like

Read More

12 Share Reply

SHOW 4 REPLIES

**derreck503** ★71 March 1, 2020 3:09 AM Report

For solution 1: Instead of doing `shape.add((r - r0) * 2 * grid[0].length + (c - c0));`, I store the coordinates as a string: `String temp = (r - r0) + "," + (c - c0);`. It's easier to read/understand IMO.

Here's my full solution:

Read More

8 Share Reply

SHOW 1 REPLY

**MockCode** ★26 October 8, 2017 9:42 PM

Can you explain how you converted our tuples (r - r0, c - c0) to integers in the Java solution? I don't quite understand.

5 Share Reply

**srishti7** ★3 March 10, 2019 10:14 AM

We can optimize the second solution and reduce space by just changing the visited indices from to 0 from 1 in the explore method.
This way we do not need to keep track of the seen matrix.

3 Share Reply

**vegito2002** ★1183 January 13, 2018 11:33 AM

Can we rely on the hash of a HashSet being unique/commensurate in terms of all its contents?

I was so obsessed with calculating a hash for `shape` in my own way, and did't realize I could have just used `shape` itself all the time.

2 Share Reply

SHOW 2 REPLIES

**vegito2002** ★1183 January 13, 2018 11:37 AM

Actually, if you were to implement a hash for `shape`, how would you do it?

I used a sum like `C1 * size_of_shape + sum (C2*x + y) for (x,y) in shape`, with different choices of C1 and C2. The OJ always manages to break my choices of constants with some large test cases in the end. I have used `31, 73` more like `997, 53` & `grid[0].length` (no

Read More

1 Share Reply

SHOW 2 REPLIES

**Geminiiiii** ★9 October 11, 2017 9:09 AM Report

Why is the time complexity O(R * C)?
Here each "add" operation should firstly check if there is a duplicate exists, then execute it. But "shape" is a hashset(arraylist) here, can we still achieve an O(1) look-up even if the element of a hashset is another type of container?

Read More

1 Share Reply

SHOW 5 REPLIES