Previous Next

711. Number of Distinct Islands II

Oct. 29, 2017 | 11.2K views

★★★☆☆ Average Rating: 2.44 (9 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 they have the same shape, or have the same shape after **rotation** (90, 180, or 270 degrees only) or **reflection** (left/right direction or up/down direction).

Example 1:

```
11000
10000
00001
00011
```

Given the above grid map, return `1`.

Notice that:

```
11
1
```

and

```
 1
11
```

are considered **same** island shapes. Because if we make a 180 degrees clockwise rotation on the first island, then two islands will have the same shapes.

Example 2:

```
11100
10001
01001
01110
```

Given the above grid map, return `2`.

Here are the two distinct islands:

```
111
1
```

and

```
1
1
```

Notice that:

```
111
1
```

and

```
1
111
```

are considered **same** island shapes. Because if we flip the first array in the up/down direction, then they have the same shapes.

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

Approach #1: Canonical Hash [Accepted]

Intuition

As in *Approach #1* to the sister problem Number of Distinct Islands, we determine local coordinates for each island.

Afterwards, we will rotate and reflect the coordinates about the origin and translate the shape so that the bottom-left-most coordinate is (0, 0). At the end, the smallest of these lists coordinates will be the *canonical representation* of the shape.

Algorithm

We feature two different implementations, but the core idea is the same. We start with the code from the previous problem, *Number of Distinct Islands*.

For each of 8 possible rotations and reflections of the shape, we will perform the transformation and then translate the shape so that the bottom-left-most coordinate is (0, 0). Afterwards, we will consider the canonical hash of the shape to be the maximum of these 8 intermediate hashes.

In Python, the motivation to use complex numbers is that rotation by 90 degrees is the same as multiplying by the imaginary unit, `1j`. In Java, we manipulate the coordinates directly. The 8 rotations and reflections of each point are `(x, y), (-x, y), (x, -y), (-x, -y), (y, x), (-y, x), (y, -x), (-y, -x)`.

Python

```python
class Solution(object):
    def numDistinctIslands2(self, grid):
        seen = set()
        def explore(r, c):
            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(complex(r, c))
                explore(r+1, c)
                explore(r-1, c)
                explore(r, c+1)
                explore(r, c-1)

        def canonical(shape):
            def translate(shape):
                w = complex(min(z.real for z in shape),
                            min(z.imag for z in shape))
                return sorted(str(z-w) for z in shape)

            ans = None
            for k in xrange(4):
                ans = max(ans, translate([z * (1j)**k for z in shape]))
                ans = max(ans,  translate([complex(z.imag, z.real) * (1j)**k
                                           for z in shape]))
            return tuple(ans)

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

        return len(shapes)
```

Java

```java
class Solution {
    int[][] grid;
    boolean[][] seen;
    ArrayList<Integer> shape;

    public void explore(int r, int c) {
        if (0 <= r && r < grid.length && 0 <= c && c < grid[0].length &&
                grid[r][c] == 1 && !seen[r][c]) {
            seen[r][c] = true;
            shape.add(r * grid[0].length + c);
            explore(r+1, c);
            explore(r-1, c);
            explore(r, c+1);
            explore(r, c-1);
        }
    }

    public String canonical(ArrayList<Integer> shape) {
        String ans = "";
        int lift = grid.length + grid[0].length;
        int[] out = new int[shape.size()];
        int[] xs = new int[shape.size()];
        int[] ys = new int[shape.size()];

        for (int c = 0; c < 8; ++c) {
            int t = 0;
            for (int z: shape) {
                int x = z / grid[0].length;
                int y = z % grid[0].length;
                //x y, x -y, -x y, -x -y
                //y x, y -x, -y x, -y -x
                xs[t] = c<=1 ? x : c<=3 ? -x : c<=5 ? y : -y;
                ys[t++] = c<=3 ? (c%2==0 ? y : -y) : (c%2==0 ? x : -x);
            }

            int mx = xs[0], my = ys[0];
            for (int x: xs) mx = Math.min(mx, x);
            for (int y: ys) my = Math.min(my, y);

            for (int j = 0; j < shape.size(); ++j) {
                out[j] = (xs[j] - mx) * lift + (ys[j] - my);
            }
            Arrays.sort(out);
            String candidate = Arrays.toString(out);
            if (ans.compareTo(candidate) < 0) ans = candidate;
        }
        return ans;
    }

    public int numDistinctIslands2(int[][] grid) {
        this.grid = grid;
        seen = new boolean[grid.length][grid[0].length];
        Set shapes = new HashSet<String>();

        for (int r = 0; r < grid.length; ++r) {
            for (int c = 0; c < grid[0].length; ++c) {
                shape = new ArrayList();
                explore(r, c);
                if (!shape.isEmpty()) {
                    shapes.add(canonical(shape));
                }
            }
        }

        return shapes.size();
    }
}
```

Complexity Analysis

- Time Complexity: $O(R*C \log{(R*C)})$, where R is the number of rows in the given `grid`, and C is the number of columns. We visit every square once, and each square belongs to at most one shape. The log factor comes from sorting the shapes.
- Space complexity: $O(R*C)$, the space used to keep track of the shapes.

Analysis written by: @awice

Rate this article: ★★★☆☆

Previous Next

Comments: 12

Sort By

Type comment here... (Markdown is supported)

Preview Post

nehasingh89 ★62 December 24, 2018 11:34 PM

Can someone please elaborate on the canonical method? its not making a lot of sense without any comments....

17 Share Reply

SHOW 1 REPLY

hehe666 ★73 January 5, 2018 1:25 AM

any reason to use `maximum of these 8 intermediate hashes`? does `maximum` have any math meanings? Thanks

5 Share Reply

SHOW 3 REPLIES

opportun1st ★9 October 1, 2018 6:07 PM

"then translate the shape so that the bottom-left-most coordinate is (0, 0)" seems confusing. A better way is to say "then translate the shape to move it to top-left as much as possible".

4 Share Reply

SHOW 1 REPLY

rexue70 ★296 August 25, 2018 1:21 PM Report

Could you please explain why you use

```
int lift = grid.length + grid[0].length;
```

Read More

3 Share Reply

timtam85 ★8 June 18, 2018 2:50 AM

@awice I can understand your hash function, but can you please explain why this hash function will guarantee working? Thanks!

3 Share Reply

zdxiq125 ★91 March 4, 2020 3:31 PM

I have tried to make some comments.

```
// input: a list of encoded coordinations of a original shape
public String canonical(ArrayList<Integer> shape) {
    String ans = "";
```

Read More

1 Share Reply

SHOW 1 REPLY

amulya123 ★38 July 19, 2018 5:55 AM

`xs[t] = c<=1 ? x : c<=3 ? -x : c<=5 ? y : -y;` how are we getting this? Can someone explain?

1 Share Reply

SHOW 2 REPLIES

Jane077 ★2 February 28, 2018 5:26 AM Report

@Chengcheng-Pei
If you see the comment, you could find below possible moves.
//x y, x -y, -x y, -x -y
//y x, y -x, -y x, -y -x

1 Share Reply

kakacharles10 ★35 March 25, 2019 12:40 PM

How do we sort 8 transformations?

0 Share Reply

vishalshah3584 ★36 July 21, 2018 7:48 AM

I didn't thought about canonical at all. Thanks for this solution.

0 Share Reply

‹ 1 2 ›