

## 305. Number of Islands II

Dec. 27, 2017 | 35.2K views

PreviousNext

★★★★★

Average Rating: 4.39 (13 votes)

A 2d grid map of  $m$  rows and  $n$  columns is initially filled with water. We may perform an *addLand* operation which turns the water at position (row, col) into a land. Given a list of positions to operate, **count the number of islands after each addLand operation**. An island is surrounded by water and is formed by connecting adjacent lands horizontally or vertically. You may assume all four edges of the grid are all surrounded by water.

**Example:**

**Input:**  $m = 3, n = 3, positions = [[0,0], [0,1], [1,2], [2,1]]$   
**Output:**  $[1,1,2,3]$

**Explanation:**

Initially, the 2d grid **grid** is filled with water. (Assume 0 represents water and 1 represents land).

```
0 0 0
0 0 0
0 0 0
```

Operation #1: addLand(0, 0) turns the water at grid[0][0] into a land.

```
1 0 0
0 0 0   Number of islands = 1
0 0 0
```

Operation #2: addLand(0, 1) turns the water at grid[0][1] into a land.

```
1 1 0
0 0 0   Number of islands = 1
0 0 0
```

Operation #3: addLand(1, 2) turns the water at grid[1][2] into a land.

```
1 1 0
0 0 1   Number of islands = 2
0 0 0
```

Operation #4: addLand(2, 1) turns the water at grid[2][1] into a land.

```
1 1 0
0 0 1   Number of islands = 3
0 1 0
```

**Follow up:**

Can you do it in time complexity  $O(k \log mn)$ , where  $k$  is the length of the **positions**?

Approach #1 (Brute force) [Time Limit Exceeded]

**Algorithm**

Reuse the code for Problem 200: [Number of Islands](#), for each *addLand* operation, just call the **numIslands** function of Problem 200 to get the number of islands after performing that operation.

```
C++JavaCopy
1 class Solution {
2 private:
3     void dfs(vector<vector<char>>& grid, int r, int c, vector<vector<bool>>& visited) {
4         int nr = grid.size();
5         int nc = grid[0].size();
6
7         if (r < 0 || c < 0 || r >= nr || c >= nc || grid[r][c] == '0' || visited[r][c]) return;
8
9         visited[r][c] = true;
10        dfs(grid, r - 1, c, visited);
11        dfs(grid, r + 1, c, visited);
12        dfs(grid, r, c - 1, visited);
13        dfs(grid, r, c + 1, visited);
14    }
15
16    int numIslands(vector<vector<char>>& grid) {
17        int nr = grid.size();
18        int nc = grid[0].size();
19
20        vector<vector<bool>> visited(nr, vector<bool>(nc, false));
21        int num_islands = 0;
22        for (int r = 0; r < nr; ++r) {
23            for (int c = 0; c < nc; ++c) {
24                if (grid[r][c] == '1' && !visited[r][c]) {
25                    ++num_islands;
26                    dfs(grid, r, c, visited);
27                }
28            }
29        }
30        return num_islands;
31    }
32 }
```

**Complexity Analysis**

- Time complexity:  $O(L \times m \times n)$  where  $L$  is the number of operations,  $m$  is the number of rows and  $n$  is the number of columns.
- Space complexity:  $O(m \times n)$  for the **grid** and **visited** 2D arrays.

Approach #2: (Ad hoc) [Accepted]

**Algorithm**

Use a **HashMap** to map index of a land to its island\_ID (starting from 0). For each *addLand* operation at position (row, col), check if its adjacent neighbors are in the **HashMap** or not and put the **island\_ID** of identified neighbors into a **set** (where each element is unique):

- if the **set** is empty, then the new land at position (row, col) forms a new island (monotonically increasing island\_ID by 1);
- if the **set** contains only one island\_ID, then the new land belongs to an existing island and island\_ID remains unchanged;
- if the **set** contains more than one island\_ID, then the new land bridges these separate islands into one island, we need to iterate through the **HashMap** to update this information (time consuming!) and decrease the number of island appropriately.

```
C++JavaCopy
1 class Solution {
2 public:
3     vector<int> numIslands2(int m, int n, vector<pair<int, int>>& positions) {
4         vector<int> ans;
5         unordered_map<int, int> land2id; // land index : island ID
6         int num_islands = 0;
7         int island_id = 0;
8         for (auto pos : positions) {
9             int r = pos.first;
10            int c = pos.second;
11            // check pos's neighbors to see if they are in the existing islands or not
12            unordered_set<int> overlap; // how many existing islands overlap with 'pos'
13            if (r - 1 >= 0 && land2id.count((r-1) * n + c)) overlap.insert(land2id[(r-1) * n + c]);
14            if (r + 1 < m && land2id.count((r+1) * n + c)) overlap.insert(land2id[(r+1) * n + c]);
15            if (c - 1 >= 0 && land2id.count(r * n + c - 1)) overlap.insert(land2id[r * n + c - 1]);
16            if (c + 1 < n && land2id.count(r * n + c + 1)) overlap.insert(land2id[r * n + c + 1]);
17
18            if (overlap.empty()) { // no overlap
19                ++num_islands;
20                land2id[r * n + c] = island_id++; // new island
21            } else if (overlap.size() == 1) { // one overlap, just append
22                auto it = overlap.begin();
23                land2id[r * n + c] = *it;
24            } else { // more than 1 overlaps, merge
25                auto it = overlap.begin();
26                int root_id = *it;
27                for (auto& kv : land2id) { // update island id
28                    kv.second = root_id;
29                }
30            }
31            ans.push_back(num_islands);
32        }
33        return ans;
34    }
35 }
```

**Complexity Analysis**

- Time complexity:  $O(L^2)$ , for each operation, we have to traverse the entire HashMap to update island id and the number of operations is  $L$ .
- Space complexity:  $O(L)$  for the **HashMap**.

P.S. C++ solution was accepted with 1409 ms runtime, but Java solution got an TLE (Time Limit Exceeded).

Approach #3: Union Find (aka Disjoint Set) [Accepted]

**Intuition**

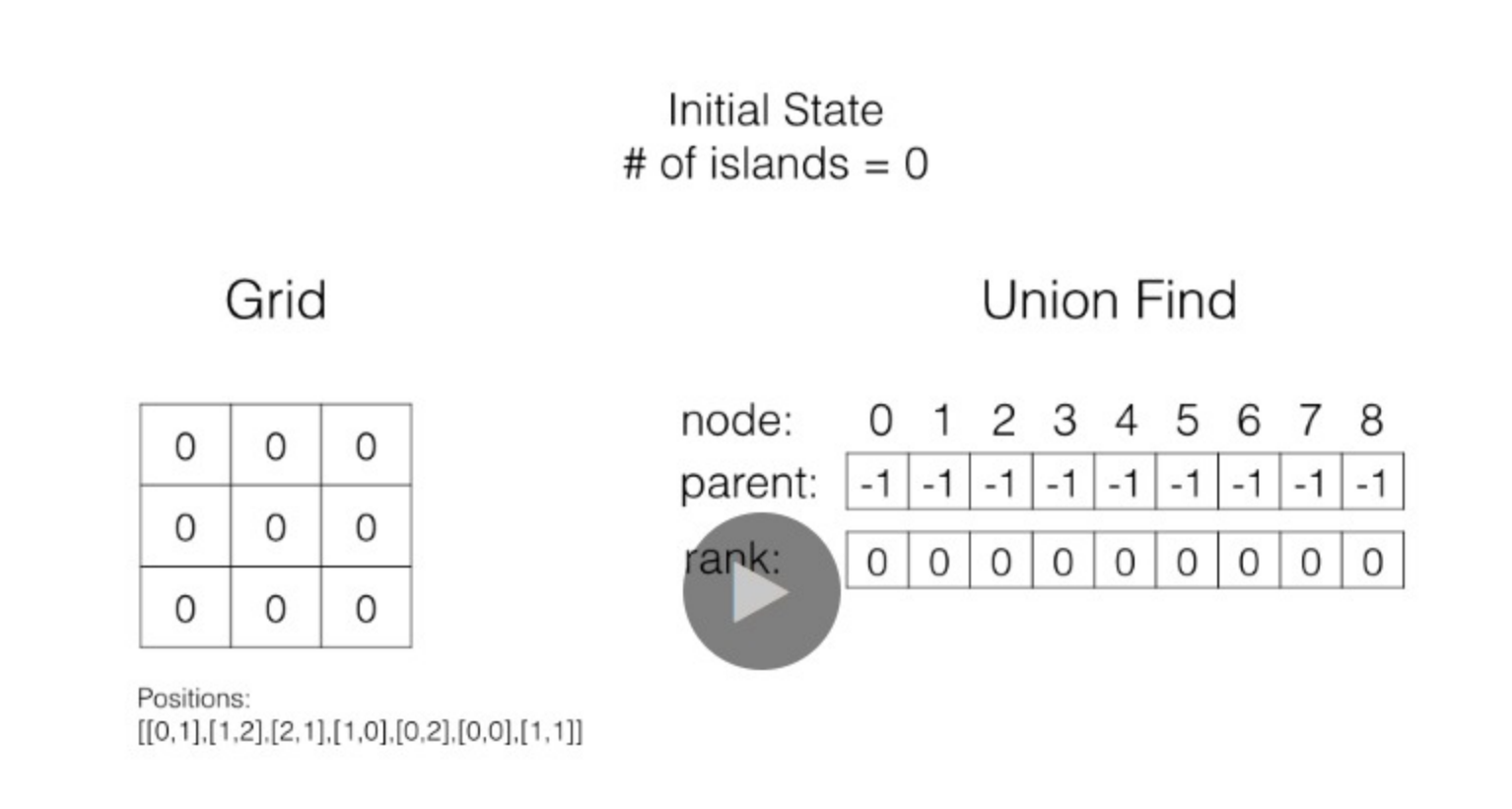
Treat the 2d grid map as an undirected graph (formatted as adjacency matrix) and there is an edge between two horizontally or vertically adjacent nodes of value **1**, then the problem reduces to finding the number of connected components in the graph after each *addLand* operation.

**Algorithm**

Make use of a **Union Find** data structure of size  $m \times n$  to store all the nodes in the graph and initially each node's parent value is set to **-1** to represent an empty graph. Our goal is to update **Union Find** with lands added by *addLand* operation and union lands belong to the same island.

For each *addLand* operation at position (row, col), union it with its adjacent neighbors if they belongs to some islands, if none of its neighbors belong to any islands, then initialize the new position as a new island (set parent value to itself) within **Union Find**.

For detailed description of **Union Find** (implemented with path compression and union by rank), you can refer to this [article](#).



1 / 15

```
C++JavaCopy
1 class UnionFind {
2 public:
3     UnionFind(int N) {
4         count = 0;
5         for (int i = 0; i < N; ++i) {
6             parent.push_back(-1);
7             rank.push_back(0);
8         }
9     }
10
11     bool isValid(int i) const {
12         return parent[i] >= 0;
13     }
14
15     void setParent(int i) {
16         parent[i] = i;
17         ++count;
18     }
19
20     int find(int i) { // path compression
21         if (parent[i] != i) parent[i] = find(parent[i]);
22         return parent[i];
23     }
24
25     void Union(int x, int y) { // union with rank
26         int rootx = find(x);
27         int rooty = find(y);
28
29         if (rootx != rooty) {
30             if (rank[rootx] < rank[rooty])
31                 parent[rootx] = rooty;
32             else if (rank[rootx] > rank[rooty])
33                 parent[rooty] = rootx;
34             else
35                 parent[rootx] = rooty;
36                 ++count;
37         }
38     }
39
40     int getCount() const {
41         return count;
42     }
43 }
```

**Complexity Analysis**

- Time complexity:  $O(m \times n + L)$  where  $L$  is the number of operations,  $m$  is the number of rows and  $n$  is the number of columns. It takes  $O(m \times n)$  to initialize UnionFind, and  $O(L)$  to process positions. Note that Union operation takes essentially constant time<sup>1</sup> when UnionFind is implemented with both path compression and union by rank.
- Space complexity:  $O(m \times n)$  as required by UnionFind data structure.

Analysis written by: @imsure.

**Footnotes**

- [https://en.wikipedia.org/wiki/Disjoint-set\\_data\\_structure](https://en.wikipedia.org/wiki/Disjoint-set_data_structure)

Rate this article: ★★★★★

Previous

Next

Comments: 22 Sort By

- Type comment here... (Markdown is supported)

PreviewPost
- vincentT

★ 29 · June 16, 2019 9:58 AM

Report

There is a bug for the third approach when we add the same position to the matrix. I suggest we can check the duplicate in the **setParent** function

Like below:

```
bool isValid(int i) {
    return parent[i] >= 0;
}
```

15 · Share · Reply

SHOW 1 REPLY
- xuwd11

★ 14 · August 8, 2018 6:50 AM

We can use a hashset to record all lands in approach 3 to avoid O(mn) initialization of disjoint set.

10 · Share · Reply

SHOW 2 REPLIES
- qja

★ 53 · August 19, 2019 4:04 AM

The Union-Find solution can be just  $O(L)$  if you don't initialize the data structure upfront and do it only on demand--in my implementation when I move to a new position in the list I initialize it (with parent and rank) in the data structure, and if a node does not exist in the parent dict it is assumed to be a zero/water.

8 · Share · Reply
- \_king\_

★ 62 · December 31, 2018 2:45 AM

```
boolean[][] visited = new boolean[nr][nc];
for (boolean[] row : visited) {
    Arrays.fill(row, false);
}
```

4 · Share · Reply
- kimjianzsu

★ 4 · February 5, 2018 9:30 PM

Report

Create parent[mn+1] and reserve parent[0] to indicate whether a cell is water. Since the init value of new int[m \* n + 1] is 0, you can avoid the array initialization of complexity O(mn). you can achieve O(klog(mn)).

4 · Share · Reply
- starfoe

★ 7 · January 2, 2020 9:58 AM

Hmmm... any solution hitting  $O(k \log(mn))$ ?

2 · Share · Reply
- dave19

★ 7 · July 1, 2019 7:33 AM

for approach #3 needs to add if (uf.isValid(index)) uf.setParent(index);

2 · Share · Reply
- jifreund

★ 2 · January 6, 2019 2:53 AM

For Solution #2, I think the complexity should be lower than what was described:

"...if the set contains more than one island\_ID, then the new land bridges these separate islands into one island. we need to iterate through the HashMap to update this information (time consuming!) and decrease the number of island appropriately."

2 · Share · Reply

SHOW 2 REPLIES
- FallingAndTrying

★ 4 · June 24, 2018 4:09 AM

One question: I think the rank will reduce after path compression. Why it was not changed?

1 · Share · Reply
- kokoss

★ 14 · August 26, 2018 11:08 PM

I have trouble understanding what the rank gives us in the Union Find solution. I know, that if we cut it, and just merge the trees so that the one with higher array index is going to be a child, that if we cut with lower array index, the results of the algorithm are not correct in some very specific (and big) cases. I wasn't able to extract a small, concise example that would fail without a rank, but pass with it. Could someone please explain to me what the rank gives us in terms of the correctness of the solution. I

0 · Share · Reply

SHOW 1 REPLY
- < 1 2 3 >