Number of Islands 4.1 (223 votes)

Number of Islands

LeetCode Admin Dec 16, 2017

Approach #1 DFS [Accepted]

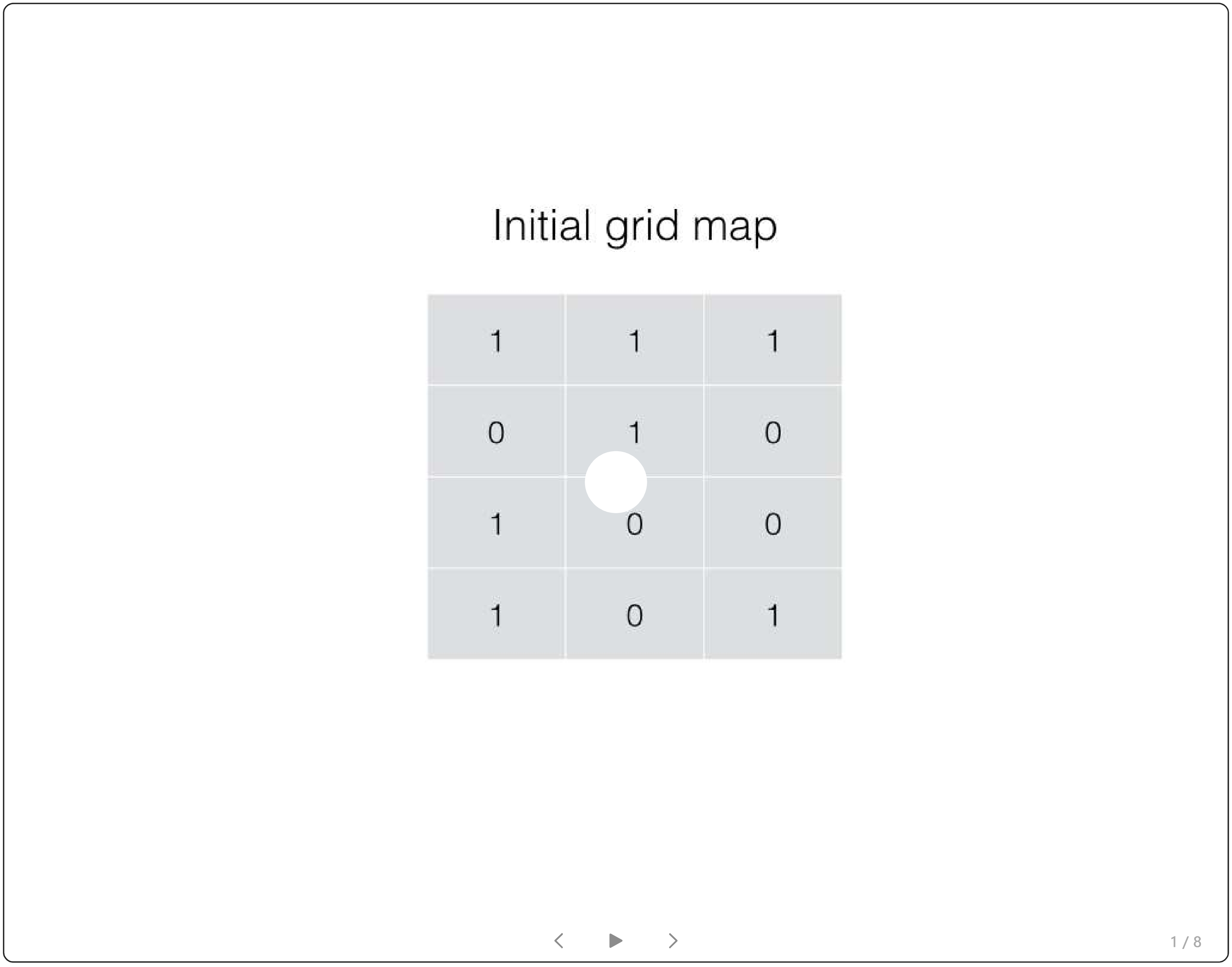
Intuition

Treat the 2d grid map as an undirected graph and there is an edge between two horizontally or vertically adjacent nodes of value '1'.

Algorithm

Linear scan the 2d grid map, if a node contains a '1', then it is a root node that triggers a Depth First Search. During DFS, every visited node should be set as '0' to mark as visited node. Count the number of root nodes that trigger DFS, this number would be the number of islands since each DFS starting at some root identifies an island.

The algorithm can be better illustrated by the animation below:



Initial grid map

1	1	1
0	1	0
1	0	0
1	0	1

C++ Java Copy

```
class Solution {
  void dfs(char[][] grid, int r, int c) {
    int nr = grid.length;
    int nc = grid[0].length;

    if (r < 0 || c < 0 || r >= nr || c >= nc || grid[r][c] == '0') {
      return;
    }

    grid[r][c] = '0';
    dfs(grid, r - 1, c);
    dfs(grid, r + 1, c);
    dfs(grid, r, c - 1);
    dfs(grid, r, c + 1);
  }

  public int numIslands(char[][] grid) {
    if (grid == null || grid.length == 0) {
      return 0;
    }

    int nr = grid.length;
    int nc = grid[0].length;
    int num_islands = 0;
    for (int r = 0; r < nr; ++r) {
      for (int c = 0; c < nc; ++c) {
        if (grid[r][c] == '1') {
```

Complexity Analysis

- Time complexity : $O(M \times N)$ where M is the number of rows and N is the number of columns.
- Space complexity : worst case $O(M \times N)$ in case that the grid map is filled with lands where DFS goes by $M \times N$ deep.

Approach #2: BFS [Accepted]

Algorithm

Linear scan the 2d grid map, if a node contains a '1', then it is a root node that triggers a Breadth First Search. Put it into a queue and set its value as '0' to mark as visited node. Iteratively search the neighbors of enqueued nodes until the queue becomes empty.

C++ Java Copy

```
class Solution {
  public int numIslands(char[][] grid) {
    if (grid == null || grid.length == 0) {
      return 0;
    }

    int nr = grid.length;
    int nc = grid[0].length;
    int num_islands = 0;

    for (int r = 0; r < nr; ++r) {
      for (int c = 0; c < nc; ++c) {
        if (grid[r][c] == '1') {
          ++num_islands;
          grid[r][c] = '0'; // mark as visited
          Queue<Integer> neighbors = new LinkedList<>();
          neighbors.add(r * nc + c);
          while (!neighbors.isEmpty()) {
            int id = neighbors.remove();
            int row = id / nc;
            int col = id % nc;
            if (row - 1 >= 0 && grid[row-1][col] == '1') {
              neighbors.add((row-1) * nc + col);
              grid[row-1][col] = '0';
            }
            if (row + 1 < nr && grid[row+1][col] == '1') {
              neighbors.add((row+1) * nc + col);
```

Complexity Analysis

- Time complexity : $O(M \times N)$ where M is the number of rows and N is the number of columns.
- Space complexity : $O(min(M, N))$ because in worst case where the grid is filled with lands, the size of queue can grow up to min(M, N).

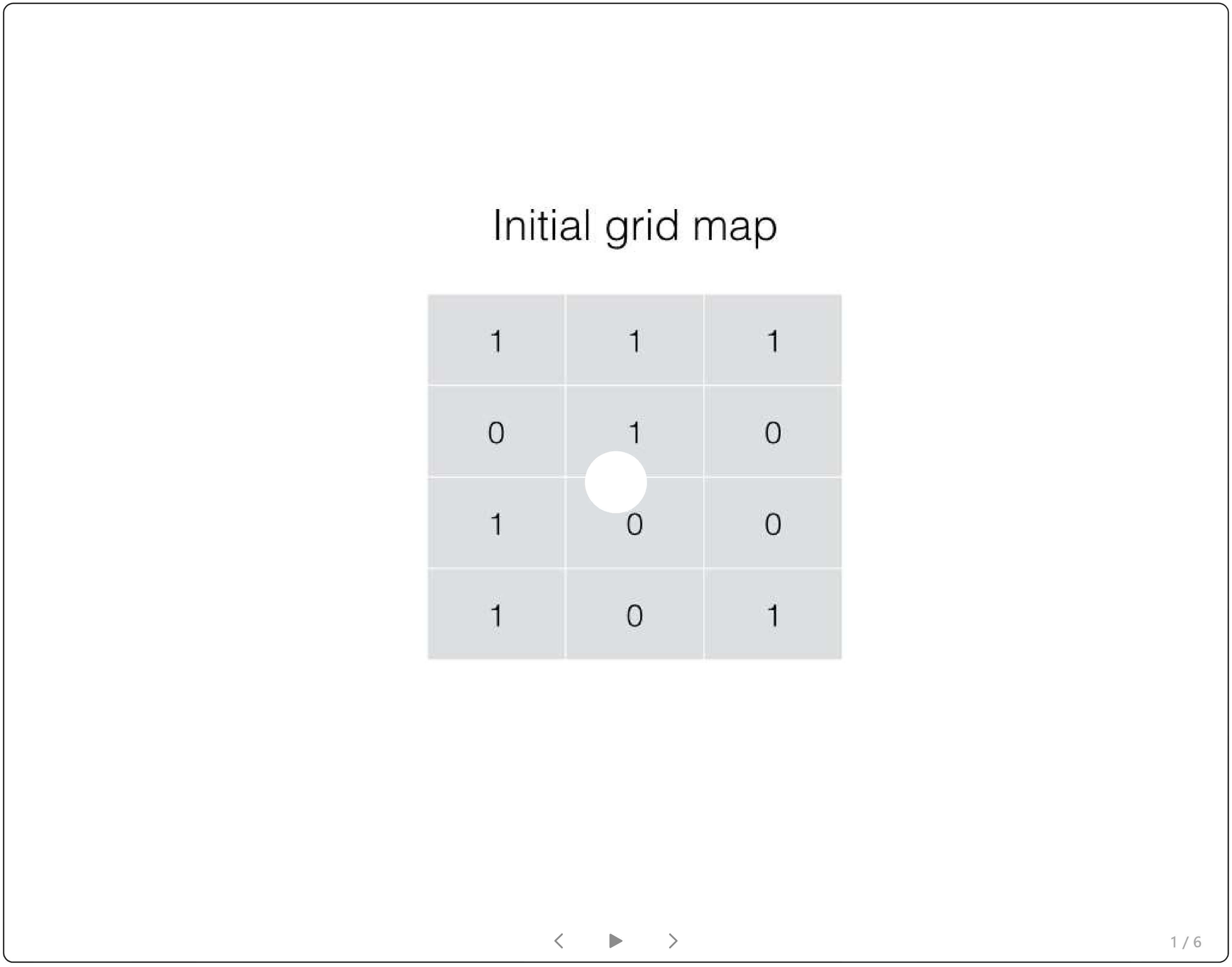
Approach #3: Union Find (aka Disjoint Set) [Accepted]

Algorithm

Traverse the 2d grid map and union adjacent lands horizontally or vertically, at the end, return the number of connected components maintained in the UnionFind data structure.

For details regarding to Union Find, you can refer to this article.

The algorithm can be better illustrated by the animation below:



Initial grid map

1	1	1
0	1	0
1	0	0
1	0	1

C++ Java Copy

```
class Solution {
  class UnionFind {
    int count; // # of connected components
    int[] parent;
    int[] rank;

    public UnionFind(char[][] grid) { // for problem 200
      count = 0;
      int m = grid.length;
      int n = grid[0].length;
      parent = new int[m * n];
      rank = new int[m * n];
      for (int i = 0; i < m; ++i) {
        for (int j = 0; j < n; ++j) {
          if (grid[i][j] == '1') {
            parent[i * n + j] = i * n + j;
            ++count;
          }
          rank[i * n + j] = 0;
        }
      }
    }

    public int find(int i) { // path compression
      if (parent[i] != i) parent[i] = find(parent[i]);
      return parent[i];
    }
```

Complexity Analysis

- Time complexity : $O(M \times N)$ where M is the number of rows and N is the number of columns. Note that Union operation takes essentially constant time[1] when UnionFind is implemented with both path compression and union by rank.
- Space complexity : $O(M \times N)$ as required by UnionFind data structure.

Footnotes

Footnotes

1. https://en.wikipedia.org/wiki/Disjoint-set_data_structure ↩

Share

Comments (201) Sort by: Best

Type comment here... (Markdown supported)

Preview Comment

💡 Article Commenting Rules

1. This comment section is for questions and comments regarding this **LeetCode article**. All posts must respect our LeetCode Community Rules.

2. Concerns about errors or bugs in the article, problem description, or test cases should be posted on LeetCode Feedback, so that our team can address them.

zoocoder_93 Apr 19, 2020

Simple attempt at understanding why the space complexity is min(M,N) in BFS.
An image is worth 1000 lines :)

https://imgur.com/gallery/M58OKvB

I hope this helps.

▲ 1.1K ▼ Show 25 Replies Reply

trickerCS Aug 29, 2019

Read more

▲ 140 ▼ Show 4 Replies Reply

prerna-p Jun 16, 2020

What is `neighbors.add(r * nc + c);` ?

▲ 56 ▼ Show 10 Replies Reply

djyale Aug 18, 2018

I'm getting Time Limit Exceeded after implementing BFS in Python with 38/47 test cases passed. Does anyone have a passing BFS solution in Python?

▲ 36 ▼ Show 14 Replies Reply

granola Oct 09, 2018

Read more

▲ 155 ▼ Show 26 Replies Reply

harryzou Dec 23, 2018

Can someone explain the validity of BFS space complexity? Thank you.

▲ 40 ▼ Show 15 Replies Reply

youryz Aug 23, 2019

I don't quite understand why the time complexity of DFS and BFS is M*N. When we iterate over the whole grid, the time complexity is MN, then for each element we again have to search its neighbors for 1s. In that way isn't the time complexity something like O(MN ** 2)?

▲ 38 ▼ Show 9 Replies Reply

ajithcherukad Apr 06, 2020

I think the space complexity is correct here for BFS approach. It would be O(min(M,N)) here for the all land case.
Example: Lets consider the below 3x4 grid. X denotes cells in the queue. The max space occupied by the queue in the below example is 3 which min(M x N)

1 1 1 1
1 1 1 1
1 1 1 1

Step 1
0 X 1 1
X 1 1 1

▲ 36 ▼ Show 5 Replies Reply

mrcslpz11 Jun 01, 2020

These solutions are bad. I did this solution in Amazon onsite and it is a bad solution. You are not allowe to wrote on the input grid, they say you are destroying the data.

▲ 72 ▼ Show 17 Replies Reply

jlwcsu Aug 17, 2020

This is not my code, but it's a working JavaScript DFS solution for those who would like an example.

Also, here's what I learned that might help others, if it isn't immediately obviuous.

I previously understood that to do DFS or BFS you needed to use recursion, in the case of you DFS you check one island, and then you use recursion on each connected island (to find the next set of bordering islands) until all the islands are found. What I did NOT understand was how to know if you had already visited an island. It was so obvious I felt like an idiot. You just delete the island. By setting the island from 1 to 0 you make sure you don't run into that island again. Thus you guarantee if you DO run into an island again it couldn't be one you already visited, because they are all now 0.

This is actually explained in the DFS solution above, but I just didn't get it. I don't know if it was the wording (since they were talking about nodes), or what deal was, but it was just unclear for me. Hope this comment helps the next person!

▲ 16 ▼ Reply

‹ 1 2 3 4 5 6 ··· 21 ›