200. Number of Islands (/problems/number-of-islands/)

Average Rating: 4.67 (3 votes)

Dec. 15, 2017 | 2.3K views

Given a 2d grid map of `'1'` s (land) and `'0'` s (water), count the number of islands. 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 1:

```
11110
11010
11000
00000
```

Answer: 1

Example 2:

```
11000
11000
00100
00011
```

Answer: 3

Credits:
Special thanks to @mithmatt (https://leetcode.com/discuss/user/mithmatt) for adding this problem and creating all test cases.

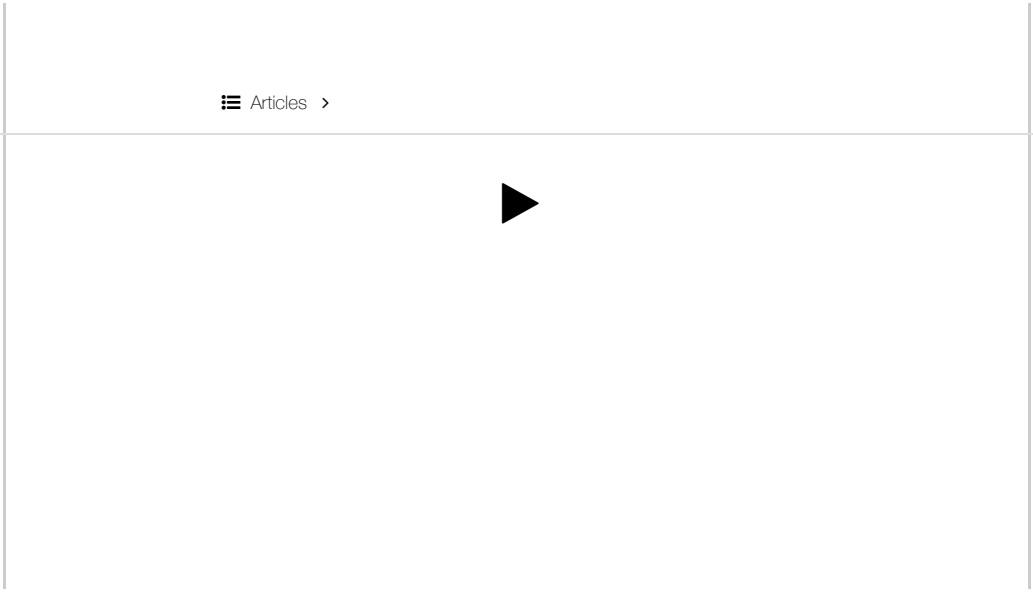
Approach #1 DFS [Accepted]

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.



1 / 8

C++ Java Copy

```
class Solution {
private:
  void dfs(vector<vector<char>>& grid, int r, int c) {
    int nr = grid.size();
    int nc = grid[0].size();

    grid[r][c] = '0';
    if (r - 1 >= 0 && grid[r-1][c] == '1') dfs(grid, r - 1, c);
    if (r + 1 < nr && grid[r+1][c] == '1') dfs(grid, r + 1, c);
    if (c - 1 >= 0 && grid[r][c-1] == '1') dfs(grid, r, c - 1);
    if (c + 1 < nc && grid[r][c+1] == '1') dfs(grid, r, c + 1);
  }

public:
  int numIslands(vector<vector<char>>& grid) {
    int nr = grid.size();
    if (!nr) return 0;
    int nc = grid[0].size();

    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;
          dfs(grid, r, c);
        }
      }
```

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(vector<vector<char>>& grid) {
    int nr = grid.size();
    if (!nr) return 0;
    int nc = grid[0].size();

    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<pair<int, int>> neighbors;
          neighbors.push({r, c});
          while (!neighbors.empty()) {
            auto rc = neighbors.front();
            neighbors.pop();
            int row = rc.first, col = rc.second;
            if (row - 1 >= 0 && grid[row-1][col] == '1') {
              neighbors.push({row-1, col}); grid[row-1][col] = '0';
            }
            if (row + 1 < nr && grid[row+1][col] == '1') {
              neighbors.push({row+1, col}); grid[row+1][col] = '0';
            }
            if (col - 1 >= 0 && grid[row][col-1] == '1') {
              neighbors.push({row, col-1}); grid[row][col-1] = '0';
```

Articles >

Complexity Analysis

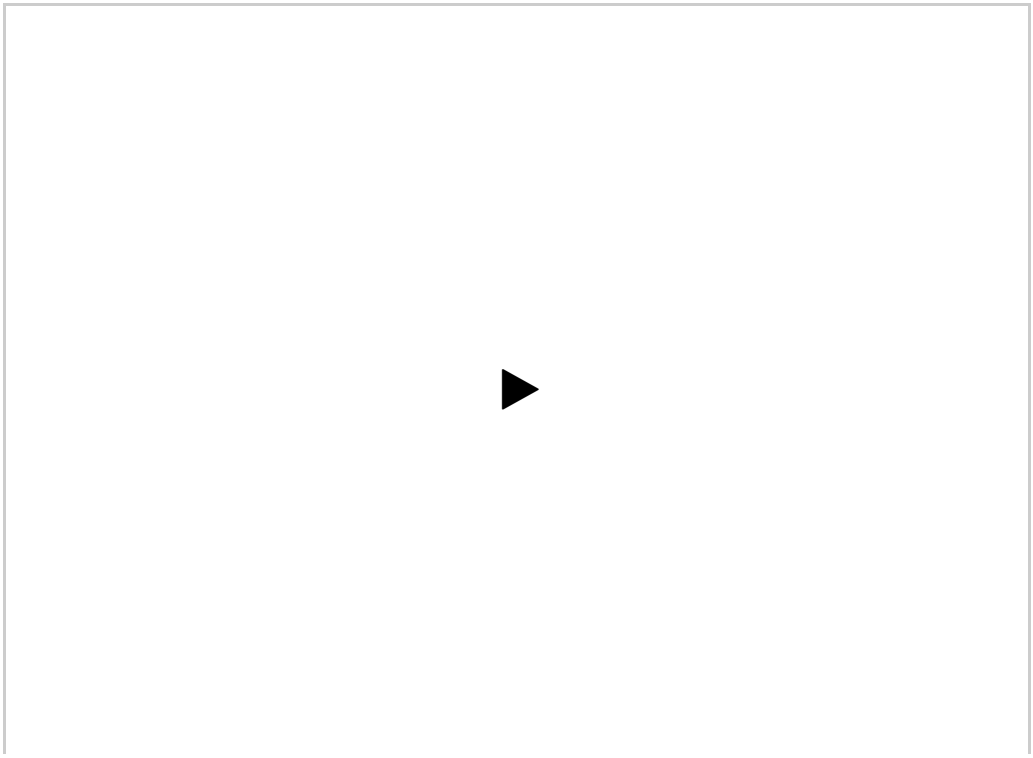
- Time complexity : $O(M \times N)$ where M is the number of rows and N is the number of columns.
- Space complexity : $O(min(M, N))$ because in worst case where the grid is filled with lands, the size of queue can grow up to min(M, N).

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 (https://leetcode.com/articles/redundant-connection/).



C++ Java

Copy

```
class UnionFind {
public:
  UnionFind(vector<vector<char>>& grid) {
    count = 0;
    int m = grid.size();
    int n = grid[0].size();
      for (int i = 0; i < m; ++i) {
        for (int j = 0; j < n; ++j) {
          if (grid[i][j] == '1') {
            parent.push_back(i * n + j);
            ++count;
          }
          else parent.push_back(-1);
          rank.push_back(0);
        }
    }
  }

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

  void Union(int x, int y) { // union with rank
    int rootx = find(x);
    int rooty = find(y);
    if (rootx != rooty) {
```

200. Number of Islands

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.

Analysis written by: @imsure (https://leetcode.com/imsure).

Thanks to @williamfu4leetcode (https://leetcode.com/williamfu4leetcode/) for correcting the space complexity analysis of BFS approach.

Footnotes

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

Rate this article: **(/ratings/107/342/?return=/articles/number-of-islands/) (/ratings/107/342/?return=/articles/numbe**

Previous (/articles/search-a-2d-matrix-ii/) Next (/articles/split-array-largest-sum/)

Join the conversation

Signed in as **tan7**.

Post a Reply

imsure commented 2 weeks ago

(https://discuss.leetcode.com/user/imsure)

@williamfu4leetcode (https://discuss.leetcode.com/uid/405622), yes you are right. I didn't think through about the space complexity of BFS approach. Thanks for making it correct! I've just updated the article to reflect this.

williamfu4leetcode commented 2 weeks ago

(https://discuss.leetcode.com/user/williamfu4leetcode)

Can you explain in your approach #2, why is space complexity O(1)? The size of queue (neighbors) will grow as more of the "newfoundland" is discovered. I have tried to run your program with a 100x100 grid of only 1 giant 100x100 island, and the size of neighbors hit 100 sometime during execution. At its peak, neighbors will contain the diagonal of the matrix, so the worst case space complexity should be min{O(M), O(N)}, which is O(M + N). (O(M) or O(N) also work, isn't it?) It's alright because we are using big O notation, I don't know how would min{O(M), O(N)} looks like in Θ, any thoughts?.

View original thread (https://discuss.leetcode.com/topic/114068)