Previous Next

200. Number of Islands

Dec. 15, 2017 | 287.6K views

Average Rating: 4.52 (68 votes)

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:

```
Input: grid = [
  ["1","1","1","1","0"],
  ["1","1","0","1","0"],
  ["1","1","0","0","0"],
  ["0","0","0","0","0"]
]
Output: 1
```

Example 2:

```
Input: grid = [
  ["1","1","0","0","0"],
  ["1","1","0","0","0"],
  ["0","0","1","0","0"],
  ["0","0","0","1","1"]
]
Output: 3
```

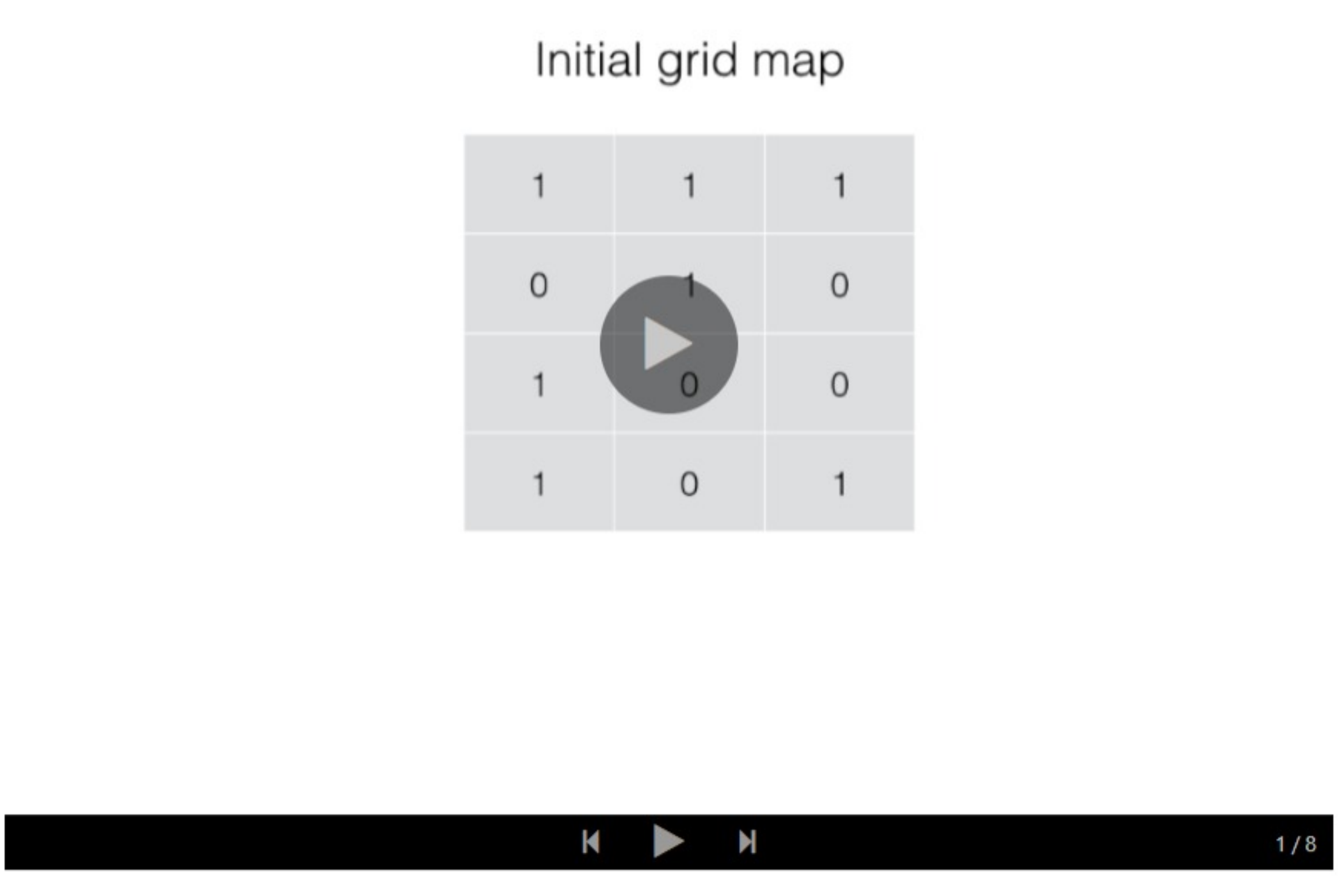
Approach #1 DFS [Accepted]

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.



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';
```

Complexity Analysis

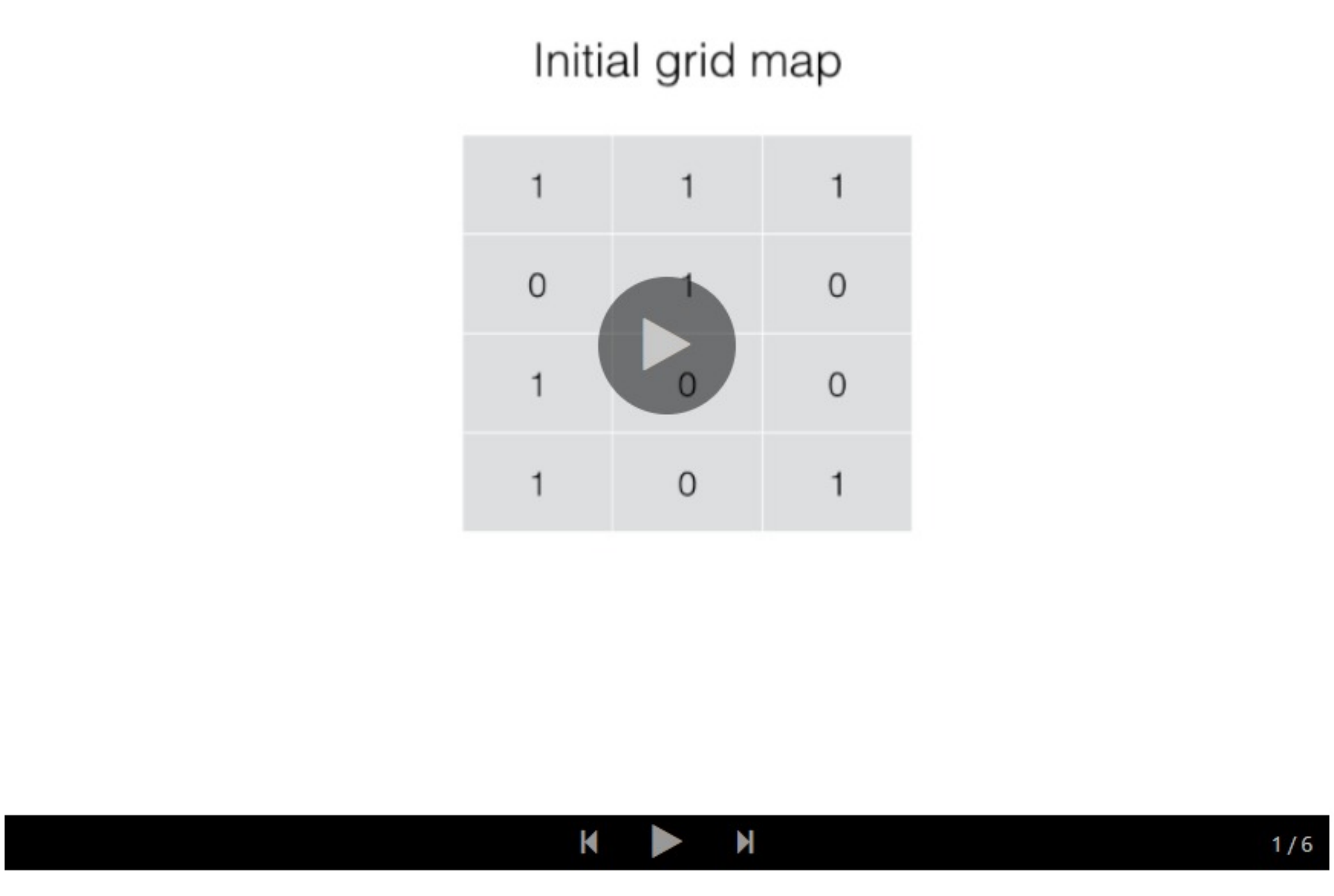
- Time complexity : $O(M \times N)$ where M is the number of rows and N is the number of columns.
- Space complexity : $O(min(M, N))$ because in worst case where the grid is filled with lands, the size of queue can grow up to min(M, N).

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.



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) {
```

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

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

Rate this article: ★★★★★

Previous Next

Comments: 66

Sort By

Type comment here... (Markdown is supported)

Preview Post

granola ★305 October 9, 2018 6:58 AM

Python solution beats 80%

```
class Solution(object):
    def numIslands(self, grid):
        islands = 0
```

Read More

67 Share Reply

SHOW 8 REPLIES

trickerCS ★33 August 28, 2019 9:55 PM

Explanation of BFS Space Complexity: Min(M, N).
Think about an example where dif(M, N) is big like 3x1000 grid. And the worst case is when we start from the middle of the grid.
Imagine how the processed points form a shape in the grid. It will be like a diamond and at some point, it will reach the longer edge of the grid. The possible shape at time t would be:

Read More

31 Share Reply

SHOW 1 REPLY

harryzou ★41 December 23, 2018 2:46 AM

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

28 Share Reply

SHOW 14 REPLIES

akashsingh ★38 April 19, 2020 7:03 AM

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

Read More

15 Share Reply

SHOW 1 REPLY

djyale ★17 August 18, 2018 7:15 AM

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?

8 Share Reply

SHOW 7 REPLIES

youryz ★15 August 22, 2019 9:13 PM

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)?

7 Share Reply

SHOW 6 REPLIES

mrcslpz11 ★4 June 1, 2020 2:40 PM

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.

4 Share Reply

SHOW 4 REPLIES

pinkfloyda ★742 February 28, 2019 12:55 PM

The union-find solution does not use path-compression. Here is a solution using that:

```
class Solution {

    int h, w;
```

Read More

3 Share Reply

SHOW 2 REPLIES

ajithcherukad ★6 April 5, 2020 9:01 PM

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)

Read More

2 Share Reply

praveen14612 ★7 May 14, 2019 7:21 AM

For #2: BFS, looks like the q grows < m+n range, so space complexity is close to O(M+N) rather than O(min(M,N)) ????

2 Share Reply

SHOW 1 REPLY

‹ 1 2 3 4 5 6 7 ›