

# Number of Islands

Solution

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:**  
11110  
11010  
11000  
00000

**Output:** 1

Example 2:

**Input:**  
11000  
11000  
00100  
00011

**Output:** 3

Java

```
1 class Solution {
2     public int numIslands(char[][] grid) {
3         int rows = grid.length;
4         if(rows == 0) return 0;
5         int cols = grid[0].length;
6         int count = 0;
7         for(int i = 0; i < rows; i++) {
8             for(int j = 0; j < cols; j++) {
9                 if(grid[i][j] == '1') {
10                    count++;
11                    grid[i][j] = '0';
12                    dfs(grid, i, j, rows, cols);
13                }
14            }
15        }
16        return count;
17    }
18 }
19
20 /** BFS */
21 private void bfs(char[][] grid, int i, int j, int rows, int cols) {
22     Queue<Integer> neighbours = new LinkedList<>();
23     neighbours.add(i * cols + j);
24     while(!neighbours.isEmpty()) {
25
26         int id = neighbours.remove();
27         int row = id / cols;
28         int col = id % cols;
29
30         // add valid neighbour to queue if not visited
31         if(row - 1 >= 0 && grid[row-1][col] == '1') {
32             neighbours.add((row-1) * cols + col);
33             grid[row-1][col] = '0';
34         }
35         if(row + 1 < rows && grid[row+1][col] == '1') {
36             neighbours.add((row+1) * cols + col);
37             grid[row+1][col] = '0';
38         }
39         if(col - 1 >= 0 && grid[row][col-1] == '1') {
40             neighbours.add((row) * cols + col-1);
41             grid[row][col-1] = '0';
42         }
43         if(col + 1 < cols && grid[row][col+1] == '1') {
44             neighbours.add((row) * cols + col+1);
45             grid[row][col+1] = '0';
46         }
47     }
48 }
```

```

46     }
47 }
48 */
49
50 /** DFS Iterative **/
51 private void dfs(char[][] grid, int i, int j, int rows, int cols) {
52     Stack<Integer> neighbours = new Stack<>();
53     neighbours.push(i * cols + j);
54     while(!neighbours.isEmpty()) {
55
56         int id = neighbours.pop();
57         int row = id / cols;
58         int col = id % cols;
59
60         // add valid neighbour to stack if not visited
61         if(row - 1 >= 0 && grid[row-1][col] == '1') {
62             neighbours.push((row-1) * cols + col);
63             grid[row-1][col] = '0';
64         }
65         if(row + 1 < rows && grid[row+1][col] == '1') {
66             neighbours.push((row+1) * cols + col);
67             grid[row+1][col] = '0';
68         }
69         if(col - 1 >= 0 && grid[row][col-1] == '1') {
70             neighbours.push((row) * cols + col-1);
71             grid[row][col-1] = '0';
72         }
73         if(col + 1 < cols && grid[row][col+1] == '1') {
74             neighbours.push((row) * cols + col+1);
75             grid[row][col+1] = '0';
76         }
77     }
78 }
79 }
80

```

☐ Custom Testcase ( [Contribute](#) )



[Run Code](#)

[Submit](#)