**QPREP6-Number of islands**

**Module Introduction**

Write a program to take a 2d grid map of 1s (land) and 0s (water) as input, count and output the number of islands.

#### Objective

Given a 2D 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.

Note: Diagonal connections don’t indicate connected land. Handle all non negative matrix sizes. For any error cases, print 0.

#### Examples

**Example 1**

Input:

**1111**0

**11**0**1**0

**11**000

00000

Output: 1

**Example 211**

Input:

**11**000

**11**000

00**1**00

000**11**

Output: 3

***SOLUTION STEPS FROM NEXT PAGE:***

**Write down at least 3 examples in the following format. Kindly stick to the format.**

**Suggestion:**

EXAMPLE#1

INPUT:

10 8

0 0 0 0 0 0 0 0

0 1 1 1 1 1 1 1

0 0 0 0 0 1 0 1

1 1 1 1 1 0 0 1

1 0 0 0 1 1 0 1

1 0 0 0 0 1 1 1

1 0 1 0 0 0 0 0

1 1 1 1 1 1 1 1

0 0 0 0 0 0 0 1

0 0 0 0 0 0 0 0

OUTPUT:

1

EXAMPLE#2

INPUT:

4 5

1 1 1 0 0

0 0 0 1 0

1 1 0 0 0

0 0 0 0 0

OUTPUT:

3

EXAMPLE#3

INPUT:

3 3

0 1 0

1 0 1

0 1 0

OUTPUT:

4

**Detail your problem understanding here**

**Suggestion:**

A matrix is provided as input with 1s and 0s.

Any combination of 1s that are connected together, either horizontally or vertically constitute a land mass. The goal is to find the number of such land masses.

Diagonal connections are not considered part of the same land mass. The boundary outside the matrix is also considered to be water. So, any land mass with 1s on the boundary should also be considered part of an island.

**Does this problem follow a known algorithmic pattern or standard application of a data structure? If there are multiple approaches, which one would you choose and why? Write down your chosen approach in 2-3 sentences like you would explain to a 10 year old.**

This problem can be treated as a graph traversal problem with Depth First Search or Breadth First Search. Pick a 1 and visit all its neighbors that are 1s and they all belong to 1 island. Start with the next 1 that is not visited and repeat.

The Union Find aka Disjoint set approach can also be used.

**Write the pseudocode here in plain English**

We will treat this matrix as a graph of nodes (with node values being ‘0’ or ‘1’) and use the Depth First Search (DFS) method to solve this problem.

class Node {

int i;

int j;

}

Read the input matrix

Initialize island count to 0

FindNumberOfIslands(Input matrix) {

Traverse through the rows and columns of the input matrix, one node at a time

If nonVisitedLandNode(), increment the island count {

doDFSAndMarkVisited(node)

}

Return island count

}

doDFSAndMarkVisited(node) {

Mark node as visited; you can store -1 or 0 to achieve that

findAllNonVisitedLandNeighbors(node) // Neighbors that are ‘1’s

for each nonVisitedLandNeighbor {

doDFSAndMarkVisited(nonVisitedLandNeighbor)

}

}

List findAllNonVisitedLandNeighbors(node) {

findAllNeighbors(node)

Add neighbors to a list if nonVisitedLandNode(neighbor) = true

Return neighborsList

}

List findAllNeighbors(node) {

Return any valid node in up, down, left, right directions

}

Boolean isValidNode(Node node) {

Return if the node indices are within matrix dimensions.

Eg: Node (-1, 0) cannot be a valid index into the matrix

}

**Can you specify a few boundary or edge cases here?**

**Edge cases**

EXAMPLE#1

INPUT:

0 0

OUTPUT:

0

EXAMPLE#2

INPUT:

1 1

1

OUTPUT:

1

EXAMPLE#3

INPUT:

3 1

0

0

1

OUTPUT:

1

EXAMPLE#4

INPUT:

1 4

0 0 0 0

OUTPUT:

0

**Write the functions you would create here**

int findNumberOfIslands(char matrix[ ][ ])

doDFSAndMarkVisited(char matrix[][], Node node) {

List findAllNonVisitedLandNeighbors(char matrix[][], Node node)

List findAllNeighbors(char matrix[][], Node node)

isValidNode(char matrix[][], Node node)

#### Summary

Starting with a brief explanation of the problem statement followed by pseudocode and then implementing the solution helps you approach the problem in a systematic way. This methodology helps with easy as well as hard problems.

**Time Complexity: O(M \* N)**

Where M is number of rows and N is number of columns

**Space Complexity: Worst case O(M \* N)**

In case that the grid map is filled with lands where DFS goes m \* n deep.

#### Concepts

Concepts covered in this Module

* Graph
* BFS
* DFS
* Union Find

Similar problems

* <https://leetcode.com/problems/surrounded-regions/>
* <https://leetcode.com/problems/walls-and-gates/>
* <https://leetcode.com/problems/number-of-connected-components-in-an-undirected-graph/>
* <https://leetcode.com/problems/max-area-of-island/>

References

* <https://www.geeksforgeeks.org/applications-of-breadth-first-traversal/>
* <https://www.geeksforgeeks.org/applications-of-depth-first-search/>
* <http://www.mathcs.emory.edu/~cheung/Courses/171/Syllabus/11-Graph/bfs.html>
* <https://www.hackerearth.com/practice/algorithms/graphs/depth-first-search/tutorial/>
* <https://stackoverflow.com/questions/2626198/graphs-data-structure-dfs-vs-bfs>
* <https://leetcode.com/articles/redundant-connection/>

### *Note*

*Note that applications of BFS & DFS are very common interview questions. So take some time and practice at least one more problem using BFS & DFS. You should be able to write iterative versions of BFS (using queues) and DFS (using stack) blindfoldedly.*

#### Good habits

Think about these for your solution:

* Comments - have you used comments in a way that others can understand this code?
* Test Cases - Are most of the scenarios/corner cases/boundary conditions handled in the solution?
* Naming Convention - Are the variables and functions named sensibly and with uniform convention?
* Modular Functions - Has the solution been addressed using concise functions? Will these functions work without any changes if they are to be used in another problem?
* Optimization - Analyze the Time Complexity and Space Complexity for your solution. Has the solution been optimized or did it use the brute force method? Is further optimization desirable/possible?
* Data Structures - Has the optimal/appropriate data structure been used?

SOLUTION:

APPROACH 1:

import java.io.\*;

import java.util.\*;

class NumberOfIslands {

int[] d = {0, 1, 0, -1, 0};

// Implement your solution by completing the below function

public int numIslands(char[][] grid) {

int islands = 0;

for (int i=0; i<grid.length; i++)

for (int j=0; j<grid[i].length; j++)

islands += sink(grid, i, j);

return islands;

}

int sink(char[][] grid, int i, int j) {

if (i < 0 || i == grid.length || j < 0 || j == grid[i].length || grid[i][j] == '0')

return 0;

grid[i][j] = '0';

for (int k=0; k<4; k++)

sink(grid, i+d[k], j+d[k+1]);

return 1;

}

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

int rows = scanner.nextInt();

int columns = scanner.nextInt();

char[][] grid = new char[rows][columns];

for (int i = 0; i < rows; ++i) {

String s = scanner.next();

for (int j = 0; j < columns; ++j) {

grid[i][j] = s.charAt(j);

}

}

scanner.close();

int result = new NumberOfIslands().numIslands(grid);

System.out.println(result);

}

}

**Complexity Analysis:**

* **Time Complexity:**
* **Space Complexity:**