Number of Islands

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 for adding this problem and creating all test cases.

Solution 1

```
public class NumberofIslands {
    static int[] dx = {-1,0,0,1};
    static int[] dy = {0,1,-1,0};
    public static int numIslands(char[][] grid) {
        if(grid==null || grid.length==0) return 0;
        int islands = 0;
        for(int i=0;i<grid.length;i++) {
            for(int j=0;j<grid[i].length;j++) {
                if(grid[i][j]=='1') {
                    explore(grid,i,j);
                    islands++;
                }
            }
        }
        return islands;
    }
    public static void explore(char[][] grid, int i, int j) {
        grid[i][j]='x';
        for(int d=0;d<dx.length;d++) {
            if(i+dy[d]<grid.length && i+dy[d]>=0 && j+dx[d]<grid[0].length && j+d
x[d]>=0 && grid[i+dy[d]][j+dx[d]]=='1') {
                explore(grid,i+dy[d],j+dx[d]);
            }
        }
    }
}
```

The algorithm works as follow:

1. Scan each cell in the grid.
2. If the cell value is '1' explore that island.
3. Mark the explored island cells with 'x'.
4. Once finished exploring that island, increment islands counter.

The arrays dx[], dy[] store the possible moves from the current cell. Two land cells ['1'] are considered from the same island if they are horizontally or vertically adjacent (possible moves (-1,0),(0,1),(0,-1),(1,0)). Two '1' diagonally adjacent are not considered from the same island.

written by fabrizio3 original link here

Solution 2

```
public class Solution {

private int n;
private int m;

public int numIslands(char[][] grid) {
    int count = 0;
    m = grid.length;
    if (m == 0) return 0;
    n = grid[0].length;
    for (int i = 0; i < n; i++){
        for (int j = 0; j < m; j++)
            if (grid[i][j] == '1') {
                DFSMarking(grid, i, j);
                ++count;
            }
    }
    return count;
}

private void DFSMarking(char[][] grid, int i, int j) {
    if (i < 0 || j < 0 || i >= n || j >= m || grid[i][j] != '1') return;
    grid[i][j] = '0';
    DFSMarking(grid, i + 1, j);
    DFSMarking(grid, i - 1, j);
    DFSMarking(grid, i, j + 1);
    DFSMarking(grid, i, j - 1);
}
```

}

written by wcyz666 original link here

Solution 3

Sink and count the islands.

Python Solution

```
def numIslands(self, grid):
    def sink(i, j):
        if 0 <= i < len(grid) and 0 <= j < len(grid[i]) and grid[i][j] == '1':
            grid[i][j] = '0'
            map(sink, (i+1, i-1, i, i), (j, j, j+1, j-1))
            return 1
        return 0
    return sum(sink(i, j) for i in range(len(grid)) for j in range(len(grid[i])))
```

Java Solution 1

```
public class Solution {
    char[][] g;
    public int numIslands(char[][] grid) {
        int islands = 0;
        g = grid;
        for (int i=0; i<g.length; i++)
            for (int j=0; j<g[i].length; j++)
                islands += sink(i, j);
        return islands;
    }
    int sink(int i, int j) {
        if (i < 0 || i == g.length || j < 0 || j == g[i].length || g[i][j] == '0'
)
            return 0;
        g[i][j] = '0';
        sink(i+1, j); sink(i-1, j); sink(i, j+1); sink(i, j-1);
        return 1;
    }
}
```

Java Solution 2

```
public class Solution {
    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;
    }
    int[] d = {0, 1, 0, -1, 0};
}
```

written by StefanPochmann original link here