#LeetCode 200

https://leetcode.com/problems/number-of-islands/description/

Yifeng Zeng

#Description

200. Number of Islands

#Idea Report

An island is all the connected points in the matrix. So we can treat a point as a node in a undrected graph, each pair of nodes next to each other has an edge connect to them. In this case, we can just traverse the whole matrix. Each time we find a point that is a part of island ('1'), we do a search from that point, and mark all the points to a special character so that in the next iterations we would skip them in order to get the number of the different islands. We can do both BFS or DFS.

When writing the code, we can ask if we need to maintain the original input, it we need, we can assign traversed island points to a special character and recover it before return.

Code

```
public class Solution {

  // BFS
  public int numIslands(char[][] grid) {
      if (grid == null || grid.length == 0 || grid[0].length == 0) {
          return 0;
      }

      int rows = grid.length;
      int cols = grid[0].length;
      int count = 0;
      for (int r = 0; r < rows; r++) {
          for (int c = 0; c < cols; c++) {
              if (grid[r][c] == '1') {
                  bfsHelper(grid, r, c);
                  count++;
```

```java
                }
            }
        }
        return count;
    }

    private void bfsHelper(char[][] grid, int row, int col) {
        int[] dx = {0, 0, 1, -1};
        int[] dy = {1, -1, 0, 0};
        Deque<int[]> q = new LinkedList<>();
        q.offer(new int[]{row, col});
        grid[row][col] = '0';

        while (!q.isEmpty()) {
            int[] cur = q.poll();
            for (int i = 0; i < dx.length; i++) {
                int r = cur[0] + dx[i];
                int c = cur[1] + dy[i];
                if (isValid(grid, r, c)) {
                    q.offer(new int[]{r, c});
                    grid[r][c] = '0';
                }
            }
        }
    }

    private boolean isValid(char[][] grid, int r, int c) {
        if (0 <= r && r < grid.length && 0 <= c && c < grid[0].length) {
            return grid[r][c] == '1';
        }
        return false;
    }
}
```

Code

```java
class Solution {

    // DFS
    public int numIslands(char[][] grid) {
        if (grid == null || grid.length == 0 || grid[0].length == 0) {
            return 0;
        }

        int rows = grid.length;
        int cols = grid[0].length;
        int count = 0;
        for (int r = 0; r < rows; r++) {
            for (int c = 0; c < cols; c++) {
                if (grid[r][c] == '1') {
                    dfsHelper(grid, r, c);
                    count++;
                }
            }
        }
        return count;
    }

    private void dfsHelper(char[][] grid, int row, int col) {
        if (row < 0 || row >= grid.length || col < 0 || col >= grid[0].length) {
            return;
        }
        if (grid[row][col] == '0') {
            return;
        }

        grid[row][col] = '0';
        dfsHelper(grid, row+1, col);
        dfsHelper(grid, row-1, col);
        dfsHelper(grid, row, col+1);
        dfsHelper(grid, row, col-1);
    }
}
```

#Summary

- Moving on a matrix can be represented as search in an undirected graph.
- Use the following array to represent the direction
    - int[] dx = {0, 0, 1, -1};
    - int[] dy = {1, -1, 0, 0};
- Use separate method isValid() to see if a move is within the boundary