Description

200. 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

Idea

An island is all the connected points in the matrix. So we can treat a point as a node in a undirected 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.

BFS

```java
public class Solution {
    // BFS AC
    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++;
                }
            }
        }
        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 ArrayDeque<>();
        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 (0 <= r && r < grid.length && 0 <= c && c < grid[0].length
                    && grid[r][c] == '1') {
                    q.offer(new int[]{r, c});
                    grid[r][c] = '0';
                }
            }
        }
    }
}
```

DFS

```java
class Solution {
    // DFS AC
    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') {
                    grid[r][c] = '0';
                    dfsHelper(grid, r, c);
                    count++;
                }
            }
        }
        return count;
    }

    private int[] dx = {0, 0, 1, -1};
    private int[] dy = {1, -1, 0, 0};
    private void dfsHelper(char[][] grid, int row, int col) {
        for (int i = 0; i < dx.length; i++) {
            int r = row + dx[i];
            int c = col + dy[i];
            if (0 <= r && r < grid.length && 0 <= c && c < grid[0].length
                && grid[r][c] == '1') {
                grid[r][c] = '0';
                dfsHelper(grid, r, c);
            }
        }
    }
}
```

Summary

- Moving in a matrix can be represented as search in an undirected graph.
- Graph coloring.
- Use the following array to represent the direction
  - int[] dx = {0, 0, 1, -1};
  - int[] dy = {1, -1, 0, 0};

C++ implementation

```
class Solution {
public:
    // BFS AC
    int numIslands(vector<vector<char>>& grid) {
        if (grid.empty()) {
            return 0;
        }
        int rows = grid.size();
        int cols = grid[0].size();
        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++;
                }
            }
        }
        return count;
    }

    void bfsHelper(vector<vector<char>>& grid, int row, int col) {
        vector<int> dx = {0,0,1,-1};
        vector<int> dy = {1,-1,0,0};
        deque<vector<int>> q;
        vector<int> cur = {row, col};
        q.push_back(cur);
        grid[row][col] = '0';
        while (!q.empty()) {
            cur = q.front();
            q.pop_front();
            for (int i = 0; i < dx.size(); i++) {
                int x = cur[0] + dx[i];
                int y = cur[1] + dy[i];
                if (0 <= x && x < grid.size() && 0 <= y && y < grid[0].size()
                    && grid[x][y] == '1') {
                    grid[x][y] = '0';
                    vector<int> next = {x, y};
                    q.push_back(next);
                }
            }
        }
    }
};
```

```cpp
class Solution {
public:
    // DFS AC
    int numIslands(vector<vector<char>>& grid) {
        if (grid.empty()) {
            return 0;
        }
        int rows = grid.size();
        int cols = grid[0].size();
        int count = 0;
        for (int r = 0; r < rows; r++) {
            for (int c = 0; c < cols; c++) {
                if (grid[r][c] == '1') {
                    grid[r][c] = '0';
                    dfsHelper(grid, r, c);
                    count++;
                }
            }
        }
        return count;
    }

private:
    vector<int> dx = {0,0,1,-1};
    vector<int> dy = {1,-1,0,0};
    void dfsHelper(vector<vector<char>>& grid, int row, int col) {
        for (int i = 0; i < dx.size(); i++) {
            int r = row + dx[i];
            int c = col + dy[i];
            if (0 <= r && r < grid.size() && 0 <= c && c < grid[0].size()
                && grid[r][c] == '1') {
                grid[r][c] = '0';
                dfsHelper(grid, r, c);
            }
        }
    }
};
```