https://course.acciojob.com/idle?question=1cc7969f-f49d-43ab-9f75-84d3921b4b4f

- MEDIUM
- Max Score: 40 Points
-

Number of Enclaves

You are given an `m x n` binary matrix grid, where 0 represents a sea cell and 1 represents a land cell.

A move consists of walking from one land cell to another adjacent (4-directionally) land cell or walking off the boundary of the grid.

Return the number of land cells in grid for which we cannot walk off the boundary of the grid in any number of moves.

Input Format

First line consists of the number of rows `m` and columns `n` of the matrix `mat`

The next `m` lines contains `n` integers each, either 0(sea) or 1(land)

Output Format

Print the number of land cells in grid for which we cannot walk off the boundary of the grid in any number of moves

Example 1

Input

```
4 4
0 0 0 0
1 0 1 0
0 1 1 0
0 0 0 0
```

Output

```
3
```

Explanation

There are three 1s that are enclosed by 0s, and one 1 that is not enclosed because its on the boundary.

Example 2

Input

```
4 4
0 1 1 0
0 0 1 0
0 0 1 0
0 0 0 0
```

Output

```
0
```

Explanation

All 1s are either on the boundary or can reach the boundary.

Constraints

```
m == grid.length

n == grid[i].length

1 <= m, n <= 500

grid[i][j] is either 0 or 1
```

Topic Tags

- Graphs

- BFS
- DFS

My code

```
// n java
import java.util.*;

public class Main {

  public static void main(String[] args) {
    Scanner sc = new Scanner(System.in);
    int m = sc.nextInt(), n = sc.nextInt();
    int[][] grid = new int[m][n];
    for (int i = 0; i < m; i++) {
      for (int j = 0; j < n; j++) {
        grid[i][j] = sc.nextInt();
      }
    }
    sc.close();
    System.out.println(numEnclaves(grid));
  }



        public static int dfs(int[][] grid, int i, int j, int m, int n,
boolean[][] vis) {
            if(i < 0 || i >= m || j < 0 || j >= n || grid[i][j] == 0 || vis[i][j]
== true)
                return 0;
```

```
            vis[i][j] = true;

            int up, down, left, right;
            // call on the up node
            up = dfs(grid, i-1, j, m, n, vis);
            // call on the down node
            down = dfs(grid, i+1, j, m, n, vis);
            // call on the left node
            left = dfs(grid, i, j-1, m, n, vis);
            // call on the right node
            right = dfs(grid, i, j+1, m, n, vis);

            return 1 + up + down + left + right;
        }

    public static int numEnclaves(int[][] grid) {
      // your code here
            int m = grid.length;
            int n = grid[0].length;

            int total = 0;

            // we calculate total 1s in the matrix
            for(int i = 0; i < m; i++) {
                    for(int j = 0; j < n; j++) {
                            if(grid[i][j] == 1) total++;
                    }
            }
```

```
boolean[][] vis = new boolean[m][n];

int count = 0;

// walk on the top border
for(int j = 0; j < n; j++) {
	if(grid[0][j] == 1 && vis[0][j] == false) {
		count += dfs(grid, 0, j, m, n, vis);
	}
}

// walk on the bottom border
for(int j = 0; j < n; j++) {
	if(grid[m-1][j] == 1 && vis[m-1][j] == false) {
		count += dfs(grid, m-1, j, m, n, vis);
	}
}

// walk on the left border
for(int i = 0; i < m; i++) {
	if(grid[i][0] == 1 && vis[i][0] == false) {
		count += dfs(grid, i, 0, m, n, vis);
	}
}

// walk on the right border
for(int i = 0; i < m; i++) {
	if(grid[i][n-1] == 1 && vis[i][n-1] == false) {
		count += dfs(grid, i, n-1, m, n, vis);
	}
```

```
        }

        return total - count;
    }



}
```