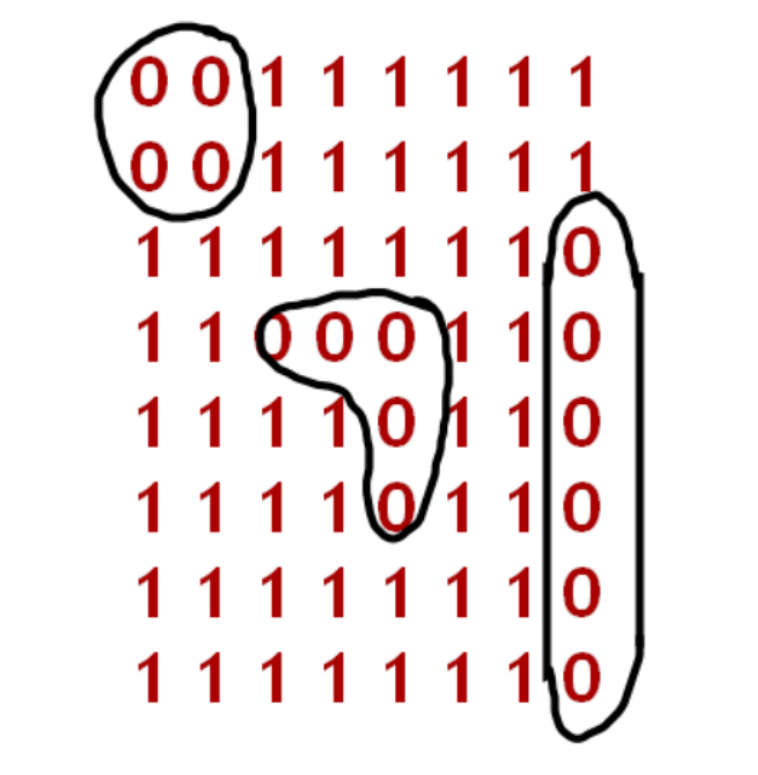
**1. Problem Discussion**

You are given a 2d array where 0's represent land and 1's represent water.Assume every cell is linked to it's north, east, west and south cell.You are required to find and count the number of islands.Reader, this problem is an application of the previously discussed question, "Get Connected Components in Graphs", so we highly request you to watch that video first, if you haven't already. Say, you are given the input 2d array as shown in figure 1.

****

For the given array, the 3 islands that can be formed are depicted in

****

**2. Approach**

Since the input is not given to us in a typical graph representation, we assume every cell of the 2d array as a vertex. We also assume that every cell is connected to its north, east, west and south cell. Hence, each of these connections is considered an edge. We first write the code for this problem and then discuss what it does.

ConsoleJava

import java.io.\*;

import java.util.\*;

public class Main {

public static void main(String[] args) throws Exception {

BufferedReader br = new BufferedReader(new InputStreamReader(System.in));

int m = Integer.parseInt(br.readLine());

int n = Integer.parseInt(br.readLine());

int[][] arr = new int[m][n];

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

String parts = br.readLine();

for (int j = 0; j < arr[0].length; j++) {

arr[i][j] = Integer.parseInt(parts.split(" ")[j]);

}

}

boolean[][]visited = new boolean[arr.length][arr[0].length]; //1

int count = 0;

for (int i = 0; i < arr.length; i++) { //2

for (int j = 0; j < arr[0].length; j++) {

if (arr[i][j] == 0 && visited[i][j] == false) {

drawTreeforComponent(arr, i, j, visited);

count++;

}

}

}

System.out.println(count); //3

}

public static void drawTreeforComponent(int[][]arr, int i, int j, boolean[][]visited) {

if (i < 0 || j < 0 || i >= arr.length || j >= arr[0].length || arr[i][j] == 1 || visited[i][j] == true) { //4

return;

}

visited[i][j] = true; //5

drawTreeforComponent(arr, i - 1, j, visited); //6

drawTreeforComponent(arr, i, j + 1, visited);

drawTreeforComponent(arr, i, j - 1, visited);

drawTreeforComponent(arr, i + 1, j, visited);

}

}

**3. CODE DISCUSSION-**

We create a new Boolean 2d array "visited" which stores whether a cell has been visited or not. Initially all the cells of this array are marked "false". We also initialize a "count" variable which stores the number of islands in the graph with 0. For every cell of the input array, we check if that cell represents "land" and it hasn't been visited before. If the cell fulfills both these conditions, then we call the drawTreeforComponent() function on it and increase the "count" by 1. The total "count" of islands in the graph is printed at the end of the main function. BASE CASE: In the drawTreeforComponent() function, if we traverse out of the input grid (i.e. row<0 or column<0 or row>=arr.length or column>=arr[0].length) or if we reach the "water" cell (arr[i][j]==1) or if we have already visited that cell earlier (visited[i][j]==true), then we simply return from the function. Before visiting a cell, we mark it as "true" so that we don't visit it again. We check whether cells to the north, east, west and south to the given cell depict the required island or not by recursively calling drawTreeforComponent() function on them.

Dear reader, in this code, we made an intelligent base case for the drawTreeforComponent() function and made "reactive" recursion calls. This means we simply make the recursion calls and if faced by any obstacle, the base case deals with it. Try running this program now. It works!

**4. Analysis-**

Time Complexity:

O(4\*n2) which is simply written as O(n2) This is because each cell of the matrix is processed at most 4 times. For Example, a particular cell can call a cell to its north, east, west or south.

Space Complexity:

O(n2) Since a 2D array is used to store "visited" elements hence the space complexity is quadratic.