

0. Presentación del Informe

Título del Informe:

Tarea Obligatoria Nro. 3: Contar islas.

Autor:

Rodrigo Ciosek.

Materia / Asignatura:

Algoritmos y Estructuras de Datos 1.

Profesor:

Carlos Mascheroni.

Fecha de entrega:

27 de mayo de 2025

1. Introducción

Desarrollar un programa en consola que resuelva la siguiente propuesta: Una matriz representa un mar e islas dentro del mismo.

- Los 0 (ceros) representan agua, y los 1 (unos) tierra.
- Una isla dentro de este mapa es un conjunto de 1 (unos) adyacentes (vertical, horizontal u oblicuamente).

Lo que se pide es una función que devuelva la cantidad de islas dado un mapa.

2. Análisis

Desarrollar una función que, dada una matriz, retorne el número de agrupaciones de valores enteros iguales a 1 que estén conectados en forma horizontal, vertical u oblicua (diagonal). Estas agrupaciones se consideran separadas por valores enteros iguales a 0.

2.1 Definición y entendimiento del problema

El problema consiste en contar la cantidad de islas en una matriz bidimensional compuesta por valores enteros 0 y 1, donde:

- El valor 0 representa agua.
- El valor 1 representa tierra.

Una isla se define como un grupo de unos (1) conectados entre sí de forma consecutiva. Las conexiones pueden darse en cualquiera de las ocho direcciones posibles: horizontal, vertical o diagonal. Es decir, un 1 forma parte de una isla si está conectado a otro 1 en una posición adyacente (arriba, abajo, izquierda, derecha o en cualquiera de las cuatro diagonales).

Dos islas están separadas si no hay una conexión directa entre sus celdas de valor 1 (están separadas por 0).

2.2 Entradas y salidas:

2.2.1 Entradas:

- Una **matriz de enteros “m”** que contiene los elementos.
- Un **índice entero “i”** que representa la posición inicial de la fila.
- Un **índice entero “j”** que representa la posición final de la columna.

2.2.2 Salidas:

- Un entero que representa el **número de agrupaciones de 1 dentro** de la matriz.

2.3 Pre y Post condiciones

2.3.1 Pre condiciones

- $m \neq \text{null}$
- $m.length > 1$
- $m[0].length > 1$
- $m[i][j] == 1 \ || \ m[i][j] == 0$

2.3.2 Post condiciones

- Retorna la cantidad de agrupaciones de números enteros 1 dentro de la matriz.
- Si la matriz no cumple con las condiciones, se retorna el valor 0.

3. Diseño

3.1 Algoritmo en lenguaje natural

Algoritmo.java

Función algoritmo

La función principal llamada algoritmo recibe como parámetro una matriz de números enteros. Su propósito es iniciar el proceso de conteo de islas dentro de la matriz.

Primero, verifica si la matriz es válida utilizando la función **validarMatriz**. Si la matriz pasa todas las validaciones, entonces llama a la función **contadorIslas** comenzando desde la posición (0, 0) de la matriz. Si la matriz no es válida, retorna 0 indicando que no es posible ejecutar el algoritmo.

Función validarMatriz

La función **validarMatriz** recibe una matriz de números enteros como parámetro y devuelve un valor booleano (verdadero o falso).

Su función es asegurarse de que la matriz cumpla con las condiciones mínimas necesarias para ejecutar el algoritmo. Las validaciones que realiza son:

- Verifica si la matriz es nula. Si lo es, retorna falso.
- Verifica si la matriz tiene al menos una fila. Si no la tiene, retorna falso.
- Verifica si la primera fila tiene al menos dos columnas. Si no las tiene, retorna falso.
- Recorre todos los elementos de la matriz para comprobar que todos sean valores 0 o 1. Si encuentra un valor distinto, retorna falso.

Si todas las validaciones se cumplen correctamente, retorna verdadero.

Función **desmarcarIsla**

La función **desmarcarIsla** recibe tres parámetros: una matriz de números enteros, un número entero que representa el índice de fila y otro que representa el índice de columna. Esta función no devuelve ningún valor.

El propósito de esta función recursiva es recorrer toda una isla (agrupación de unos) y marcar cada celda de la misma como 0, para indicar que ya ha sido visitada y no debe contarse de nuevo.

Primero, se evalúan las condiciones de corte para finalizar la recursividad:

- Si el índice de la fila es menor que 0.
- Si el índice de la columna es menor que 0.
- Si el índice de la fila es mayor o igual al número total de filas de la matriz.
- Si el índice de la columna es mayor o igual al número total de columnas de la fila.
- Si el valor de la celda actual es 0.

Si ninguna de estas condiciones se cumple, significa que se encontró una celda válida con valor 1. Esta celda se marca con valor 0, y luego se llama recursivamente a la función **desmarcarIsla** para las ocho direcciones posibles: arriba, abajo, izquierda, derecha, arriba derecha, arriba izquierda, abajo derecha y abajo izquierda.

Este proceso se repite mientras se encuentren valores 1 a su alrededor, marcando cada celda visitada como 0 hasta recorrer por completo la isla. Luego de que no se encuentren mas valores 1, este vuelve desacoplando todas las funciones guardadas en la pila, volviendo hasta su ubicación que fue ejecutada.

Función **contadorIslands**

La función **contadorIslands** recibe como parámetros una matriz de enteros, un índice de fila y un índice de columna. Devuelve un número entero que representa la cantidad de islas encontradas.

Su función es recorrer la matriz entera de forma recursiva, buscando celdas con valor 1. Cada vez que encuentra una celda con valor 1, la considera como el inicio de una nueva isla, llama a la función **desmarcarIsla** para marcar esa isla como visitada, y suma 1 al contador de islas.

El funcionamiento se puede resumir en los siguientes pasos:

- Si el índice de la fila es igual al número total de filas, significa que se llegó al final de la matriz, por lo tanto retorna 0.
- Si el índice de la columna alcanza el final de la fila actual, la función se vuelve a llamar con la siguiente fila y la columna reiniciada en 0.
- Si se encuentra una celda con valor 1, se considera una nueva isla. Se llama a la función **desmarcarIsla** para marcar esa isla como visitada. Luego, se vuelve a iniciar el recorrido desde la posición (0, 0), y se suma 1 al resultado.
- Si la celda actual no cumple ninguna de estas condiciones, se continúa el recorrido hacia la siguiente columna.

Este proceso garantiza que cada isla se cuente una única vez, sin importar su forma o posición dentro de la matriz.

4. Implementación

Algoritmo.java

```
public class Algoritmo { 1 usage  ⚙️ Rodrigo *

    public static int algoritmo(int[][] m){ 1 usage  new *
        if (validarMatriz(m) ){
            return contadorIslas(m, 0, 0);
        }
        return 0;
    }

    public static int contadorIslas(int[][] m, int i, int j) { 4 usages  new *
        if (i == m.length) return 0;
        if (j == m[i].length) return contadorIslas(m, i + 1, 0);

        // Si hay tierra
        if (m[i][j] == 1) {
            desmarcarIsla(m, i, j);

            return 1 + contadorIslas(m, i, j);
        }
        return contadorIslas(m, i, j + 1);
    }


    public static void desmarcarIsla(int[][] m, int i, int j) { 9 usages  ⚙️ Rodrigo *
        if (i < 0 || j < 0 || i >= m.length || j >= m[i].length || m[i][j] == 0)
            return;

        m[i][j] = 0; // Marcamos 0 para indicar visitada

        desmarcarIsla(m, i + 1, j); // Arriba
        desmarcarIsla(m, i - 1, j); // Abajo
        desmarcarIsla(m, i, j + 1); // Derecha
        desmarcarIsla(m, i, j - 1); // Izquierda
        desmarcarIsla(m, i + 1, j + 1); // Abajo derecha
        desmarcarIsla(m, i + 1, j - 1); // Abajo izquierda
        desmarcarIsla(m, i - 1, j + 1); // Arriba derecha
        desmarcarIsla(m, i - 1, j - 1); // Arriba izquierda
    }

    public static boolean validarMatriz(int[][] m){ 1 usage  new *
        if (m == null) return false;
        if (m.length < 1) return false;
        if (m[0].length < 2) return false;
        for (int i = 0; i < m.length; i++) {
            for (int j = 0; j < m[i].length; j++) {
                if (m[i][j] != 0 && m[i][j] != 1) return false;
            }
        }
        return true;
    }
}
```


Main.java

public class Main {  Rodrigo *

 1 ^

```
static int[][] isla = { no usages
    {0, 0, 0, 1, 1, 0, 0, 0, 1, 0},
    {0, 0, 1, 0, 1, 0, 0, 0, 1, 1},
    {0, 1, 0, 0, 1, 1, 1, 0, 0, 0},
    {0, 1, 0, 0, 0, 0, 0, 1, 0, 0},
    {0, 0, 1, 0, 1, 0, 1, 0, 0, 1},
    {0, 0, 1, 0, 0, 0, 1, 0, 0, 1},
    {0, 0, 1, 0, 0, 0, 1, 0, 0, 1},
    {0, 0, 0, 1, 1, 1, 0, 0, 1, 0},
    {0, 0, 0, 0, 0, 0, 0, 1, 0, 0},
    {0, 0, 0, 0, 0, 0, 1, 0, 0, 0},
    {0, 0, 1, 1, 0, 0, 1, 0, 1, 0},
    {0, 0, 0, 0, 0, 0, 0, 0, 0, 1}
};

static int[][] isla2 = {}; 1 usage

public static void main(String[] args) {  Rodrigo *
    System.out.println(Algoritmo.algoritmo(isla2));
}
```

5. Verificación y Validación

Pruebas

- Caso básico 1:
 - {
 {0, 0, 0, 1, 1, 0, 0, 0, 1, 0},
 {0, 0, 1, 0, 1, 0, 0, 0, 1, 1},
 {0, 1, 0, 0, 1, 1, 1, 0, 0, 0},
 {0, 1, 0, 0, 0, 0, 0, 1, 0, 0},
 {0, 0, 1, 0, 1, 0, 1, 0, 0, 1},
 {0, 0, 1, 0, 0, 0, 1, 0, 0, 1},
 {0, 0, 1, 0, 0, 0, 1, 0, 0, 1},
 {0, 0, 0, 1, 1, 1, 0, 0, 1, 0},
 {0, 0, 0, 0, 0, 0, 0, 1, 0, 0},
 {0, 0, 0, 0, 0, 0, 1, 0, 0, 0},
 {0, 0, 1, 1, 0, 0, 1, 0, 1, 0},
 {0, 0, 0, 0, 0, 0, 0, 0, 0, 1}
} → Resultado esperado: 6
- Caso básico 2:
 - {
 {0, 0, 0, 1, 1, 0, 0, 0, 1, 0},
 {0, 0, 1, 0, 1, 0, 0, 0, 1, 1},
 {0, 1, 0, 0, 1, 1, 1, 0, 0, 0},
 {0, 1, 0, 0, 0, 0, 0, 1, 0, 0},
 {0, 0, 1, 0, 1, 0, 1, 0, 0, 1},
 {0, 0, 1, 0, 0, 0, 1, 0, 0, 1},
 {0, 0, 1, 0, 0, 0, 1, 0, 0, 1},
 {0, 0, 0, 1, 1, 1, 0, 0, 1, 0},
 {0, 0, 0, 0, 0, 0, 0, 1, 0, 0},
 {0, 0, 0, 0, 0, 0, 1, 0, 0, 0},
 {0, 0, 1, 1, 0, 0, 1, 0, 1, 0},
 {0, 0, 0, 0, 0, 0, 0, 0, 0, 2}
} → Resultado esperado: 0
- Matriz vacía: {} → Resultado esperado: 0

Resultados:

- Caso básico 1: Retorna 6 (Correcto)
- Caso básico 2: Retorna 0 (Correcto)
- Matriz vacía: 0 (Correcto)

6. Conclusiones

Tuve muchos intentos en resolver este ejercicio, primero se me ocurrió recorrer columna por columna e ir guardando los índices que en la siguiente fila podrían estar

unidos a los 1 de la actual e ir sumando las islas actuales, pero luego me di cuenta que no iba a funcionar, ya que capaz que los 1 se conectan en un futuro por otros 1 y yo ya sumé las islas.

Luego hice este algoritmo con el que iba detectando un uno y me metía a desmarcar islas y volvía al principio hasta que el algoritmo llegara al final. Lo resolví, pero al principio me pasó en la función de desmarcar islas, que no recorría todas las posiciones en todos los casos posibles, entonces no me desmarcaba todas las tierras, entonces me formaba islas nuevas creadas por mí, y bueno, al final me di cuenta que sí o sí tenía que recorrer todas las posiciones en todos los casos.