PREGUNTAS SECCIÓN 4

Sesión práctica 7

Jonathan Arias Busto
UO283586
71780982-Y
Escuela de Ingeniería Informática - EII

CUESTIONES

1. Explica cómo funciona tu algoritmo.

A la hora de hacer el algoritmo hay que tener en cuenta varias cosas como por ejemplo la forma en la que tenemos los datos y como los almacenamos. En este caso tenemos 2 ficheros en formato .txt que nos dan los diferentes países de Europa con sus respectivas fronteras. En el otro fichero tenemos los diferentes colores que podrán ser usados para pintar.

El algoritmo en sí trata de colorear el mapa de Europa. Este problema se trata del típico teorema de coloreado de grafos.

En mi caso el algoritmo va obteniendo los diferentes países almacenados en un HashMap para el cual la clave es el país y el valor es una lista que contiene las diferentes fronteras para dicho país. Con esta información podemos buscar un color para dicho país que no coincida con ningún color de la frontera. Para poder hacer estos hay varios métodos auxiliares que se encargan de esto. También me ayudo de otro HashMap para el cual la clave es el país y el valor es el color que contiene. Con esto lo que conseguimos es ir almacenando los colores para los distintos países y poder ir así eligiendo los distintos colores.

El método principal del algoritmo es el siguiente:

```
public HashMap<String, String> coloracionMapa() {
    HashMap<String, String> res = inicializarRes(); // Pais con color respectivo
    for (String pais : fronteras.keySet()) {
        if (res.get(pais) == "empty") {
            String colorElegido = elegirColor(fronteras.get(pais), res);
            res.put(pais, colorElegido);
        }
    }

    return res;
}
```

Se puede ver el HashMap<Pais, color>, antes nombrado, en la primera linea de código del método. Este HashMap se inicializa con los valores = “empty”, para así poder distinguir entre los países que tienen un color asignado y otro que no.
El siguiente paso es ir iterando a través del HashMap<Pais, Fronteras> y obtener el color solución para ese país. La obtención de dicho color se explicará más adelante.
Y por último se introduce en el HashMap<País, Color> la información.

Ahora vamos a ver como se obtiene el color para cada país. Para llevar a cabo esto fue necesario la utilización de 3 métodos, 1 el cual es auxiliar.
Comenzaremos por el elegirColor() que es el primero al que se le hace la llamada. Este método tiene el siguiente aspecto:

```
private String elegirColor(List<String> frontera, HashMap<String, String> paises) {
    String aux = colores[0];
    if (frontera.size() == 1 && frontera.get(0).trim().equals("NO")) {
        return colores[0];
    }
    ArrayList<String> colAux = new ArrayList<String>();
    for (String f : frontera) {
        String color = paises.get(f.trim());
        if (color != "empty") {
            colAux.add(color);
        }
    }
    if (colAux.size() == 0) {
        aux = colores[0];
    } else {
        aux = searchColor(colAux);
    }
    return aux;
}
```

Este método recibe como parametros el HashMap<Pais, Color> y una lista con las diferentes fronteras de un pais. Lo que va haciendo dicho método es iterar a través de los diferentes países frontera y comprobar si ya tiene un color, si ya tiene un color se añade a una lista auxiliar donde estarán todos los colores distintos que tienen las fronteras de un país. Cuando se acaba de iterar se hace una llamada al método searchColor() que será el que nos darña el color solución para el país.

Este método tiene el siguiente aspecto:

```
private String searchColor(ArrayList<String> colAux) {
    int[] aux = new int[colAux.size()];
    for (int i=0;i<colAux.size();i++) {
        aux[i] = getColorIndex(colAux.get(i));

    }
    Arrays.sort(aux);
    int cont = 0;
    for (int i=0;i<aux.length;i++) {
        if (cont == aux[i]) cont++;
    }

    return colores[cont];
}
```

Este método lo primero de todo lo que hace con la lista de colores pasada como parámetro es convertirlo en un array de enteros con los indices de los colores para el vector de colores cargado del .txt. Para hacer esto se ayuda del metodo getColorIndex() que recibe como parámetro un color en forma de String y devuelve la posición para el array de colores.

```
private int getColorIndex(String color) {
    int count = 0;
    for (String c : colores) {
        if (c == color) return count;
        else count++;
    }
    return -1;
}
```

Una vez tenemos dicho array de posiciones lo que hacemos es ordenarlo a través de la orden Arrays.sort() y lo que tenemos que hacer para obtener el color solución es comprobar la posición del primer color libre comparándolo con el array de colores auxiliar calculado anteriormente.

Con este índice del color podemos elegir el color y así meterlo en el HashMap<País, Color>.

Este es el funcionamiento por encima del algoritmo creado para solucionar el problema.

2. ¿Cuántos colores han sido necesitados para resolver el problema dado?

En mi caso han sido necesarios 5 colores (Rojo, Azul, Verde, Amarillo y Negro).

3. ¿Podría cambiarse el número de colores necesarios si se utilizase un orden diferente para procesar los datos de entrada?

Si se podría mejorar, en mi opinión, obteniendo el HashMap de paises y fronteras de una forma ordenada, es decir, por ejemplo:

1. Obtener Portugal.
2. Obtener España.
3. Obtener Andorra.
4. Obtener Francia.

Así hasta el último país. Con esto quiero decir que sería mejor, reflexionando, empezar con un país colocado en un extremo y continuar el algoritmo desde la frontera de este país e ir avanzado así a través de todo el mapa de Europa.

4. ¿Cuántos colores utilizarías, como mucho, en una solución optima?

Utilizaría 4 colores como mucho, el problema es que obtener este algoritmo es muy complicado, ya que es un problema NP.

5. ¿Cual es la complejidad temporal de tu algoritmo?

El algoritmo implementado en teoría tiene una complejidad cuadrática. No hay forma de medir tiempos para comprobarlo puesto que este problema tiene siempre la misma carga al cargar los países de Europa y esta carga es siempre constante.

Mirando el código podemos ver como el método principal tiene un for y dentro una llamada a un método que tiene complejidad O(n), por tanto O(n^2) sería la complejidad de dicho algoritmo.