Universidad Nacional de La Matanza
Departamento de Ingeniería e Investigaciones Tecnológicas

Programación Avanzada

Trabajo Práctico Nº1:
Procesando imágenes

Comisión: Jueves Tarde

Grupo: Beta

Tema: Inversión de colores

Fecha de Presentación: 05/09/2024

Integrantes:

DNI	Apellido	Nombre	Email
43630151	Antonioli	Iván Oscar	iantonioli@alumno.unlam.edu.ar
43664669	Di Nicco	Luis Demetrio	ldinicco@alumno.unlam.edu.ar

Índice:

Introducción

En este Trabajo práctico investigaremos sobre el procesamiento de imágenes mediante la aplicación de diversos algoritmos sobre matrices de enteros.

Específicamente, profundizaremos sobre el algoritmo utilizado para convertir una imagen con formato PGM en su negativo invirtiendo los valores de intensidad de los píxeles. Durante el desarrollo de este trabajo, implementamos el algoritmo en java y experimentamos con distintas imágenes de prueba para conocer cómo se comporta la inversión de colores en cada caso.

Desarrollo

¿Cómo funciona el formato PGM?

El formato de imagen PGM (Portable Gray Map) es un formato de archivos que almacena imágenes en escala de grises. Cada píxel de la imagen se representa por un valor de brillo, donde 0 equivale al color negro y el valor máximo equivale al color blanco.

Tipos de Formato

Un archivo PGM puede estar en uno de los siguiente dos tipos de formatos:

1. **P2 (ASCII)**: Los valores de los píxeles se guardan como texto legible, lo que hace que el archivo sea más fácil de leer pero más grande en tamaño.
2. **P5 (Binario)**: Los valores de los píxeles se almacenan en formato binario, lo que resulta en archivos más compactos y rápidos de procesar.

Ambos formatos comienzan con una cabecera que incluye información sobre el tipo de archivo, las dimensiones de la imagen y el valor máximo de gris, esta información está almacenada en ASCII.

La primera línea indica el tipo de formato del archivo pgm (P2 o P5), la segunda línea indica las dimensiones de la imagen, el primer valor corresponde al ancho de la imagen y el segundo valor a la altura. Finalmente, el encabezado indica el valor máximo que puede tomar un píxel en la imagen.

Ejemplo:

```
P5
274 184
255
```

La imagen corresponde a un archivo .pgm formato P5, por lo tanto, los píxeles están almacenados en binario. La imagen tiene un ancho de 274 píxeles y 184 píxeles de altura. El máximo valor que puede tomar un píxel en este archivo es de 255.

¿Cómo funciona el algoritmo de Inversión de Colores?

Algoritmo inversión de Colores mediante complemento al valor máximo

La inversión de color en una imagen en escala de grises es un proceso que transforma cada píxel al valor opuesto en la escala de grises. Para obtener el valor inverso u opuesto del píxel se debe obtener su complemento, es decir, tomar el valor máximo de la escala de grises leído del encabezado del archivo y restarle el valor actual del píxel. La transformación para obtener el inverso de un pixel se puede resumir en la siguiente fórmula:

$$Valor\ Invertido\ =\ Valor\ máximo\ -\ Valor\ Actual\ Pixel$$

Ejemplo:

- Si el valor original de un píxel es 100, y el valor máximo en la imagen es 255, entonces el nuevo valor invertido se calcula de la siguiente manera:

$$255\ -\ 100\ =\ 150$$

Características de la inversión:

- Un píxel negro (valor 0) se convierte en blanco (valor 255).
- Un píxel blanco (valor 255) se convierte en negro (valor 0).
- Un píxel con un valor intermedio se convierte en su valor complementario en la escala de grises.

Este proceso se aplica a cada píxel de la imagen para obtener la imagen con los colores invertidos. Se debe recorrer la matriz de enteros que representa los valores de los píxeles en su totalidad para poder obtener la imagen con los colores invertidos.

Ejemplo transformación de imágenes utilizando este algoritmo:

Imagen Original *Imagen Invertida*

Se puede apreciar que los tonos oscuros de la imagen se convirtieron en claros y viceversa, esto permite observar nuevos detalles y características que en la imagen original son difíciles de percibir.

Algoritmo inversión de Colores mediante complemento al valor real

Otra forma de aplicar el algoritmo de inversión de colores es aplicando una inversión pero en vez de hacerlo respecto al valor máximo proporcionado por el archivo, se hace respecto al valor máximo real de la imagen, el cual se obtiene luego de recorrer toda la matriz. Esta variación permite resaltar aún más las pequeñas variaciones entre píxeles, sacando a la luz detalles que con el algoritmo anterior pasaría desapercibido.

Para aplicar este algoritmo se deben realizar dos pasos. Primero se debe calcular el valor escalado del píxel actual, el cual se obtiene dividiendo el valor máximo del archivo en "segmentos" del tamaño del valor máximo real y multiplicando ese resultado por el valor del píxel actual. Luego de obtener el valor escalado del pixel, el segundo paso es realizar la inversión de la misma forma en la que se hacía en el algoritmo anterior.

$$Valor\ Escalado\ Pixel\ =\ (Valor\ máximo\ Archivo/Valor\ Maximo\ Real)\ *\ Valor\ Actual\ Pixel$$

$$Valor\ Invertido\ =\ Valor\ máximo\ Archivo\ -\ Valor\ Escalado\ Pixel$$

Ejemplo:

- Si el valor máximo proporcionado por el archivo es 255, pero el valor máximo real del archivo es 20, y el valor actual del pixel es 5, el valor invertido será:

$$Valor\ Escalado\ Pixel\ =\ (Valor\ máximo\ Archivo/Valor\ Maximo\ Real)\ *\ Valor\ Actual\ Pixel$$

$$Valor\ Escalado\ Pixel\ =\ (255\ /\ 20)\ *\ 5\ =\ 63.75 \cong 63$$

$$Valor\ Invertido\ =\ Valor\ máximo\ Archivo\ -\ Valor\ Escalado\ Pixel$$

$$Valor\ Invertido = 255\ -\ 63.75\ =\ 191.25\ \cong 191$$

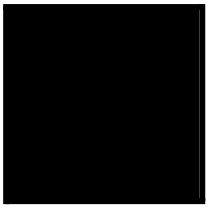
Utilizando el algoritmo original el valor del pixel hubiese sido 250 (255 - 5 = 250). La imagen hubiese quedado casi toda en blanco y no se hubiesen podido distinguir las diferencias entre los valores de los píxeles. Con esta alternativa, se pueden distinguir mejor las diferencias y notar detalles que a simple vista o luego de aplicar el primer algoritmo explicado no se verían.

Comparación entre ambos algoritmos

Por ejemplo, en la siguiente imagen el valor máximo proporcionado por el archivo es 255, sin embargo el valor máximo real de la imagen es 1. Originalmente esta imagen se verá en su totalidad de color negro, y al aplicar el algoritmo de inversion, pasará a verse en su totalidad de color blanco, siendo imperceptible la diferencia entre los pixeles con valor original de 1 respecto a los que originalmente tenian un valor de 0. Aplicando esta alternativa del algoritmo se consigue resaltar la diferencia entre los valores de los píxeles.

- Imagen Original

```
P2
3 3
255
0 0 0
1 1 1
0 0 0
```



Originalmente, a simple vista la imagen parece estar completamente en negro.

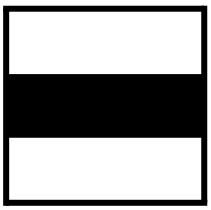
- Imagen invertida respecto al máximo proporcionado por el archivo (255)

```
P2
3 3
255
255 255 255
254 254 254
255 255 255
```

Luego de aplicar el algoritmo de inversión de colores mediante el complemento del valor Maximo, la imagen a simple vista parece estar completamente en blanco. En este caso, no fue de mucha ayuda aplicar esta transformación.

- Imagen invertida respecto al máximo real del el archivo (1)

```
P2
3 3
255
255 255 255
  0   0   0
255 255 255
```



Luego de aplicar el algoritmo de inversión de colores mediante el complemento del valor máximo real, se resaltan mejor las diferencias entre los valores de los píxeles.

Análisis Teórico de la Complejidad Computacional del Algoritmo

Clases del Algoritmo

Para desarrollar el algoritmo solicitado dividimos el proyecto en las siguientes clases:

- EscribirPGM: Contiene las funciones enfocadas en crear los archivos de Salida en formato P2 o P5.
- LeerPGM: Contiene las funciones enfocadas en leer los archivos de Entrada en formato P2 o P5.
- ImagenPGM: Contiene las funciones y la lógica encargada de realizar la inversión del color así como toda la información correspondiente a la imagen.
- MostrarImagen: Contiene las funciones dedicadas a mostrar las imágenes en JAVA
- PanelImagen:Contiene las funciones dedicadas a mostrar las imágenes en JAVA

Análisis de Complejidad de Función de lectura de Imágenes PGM:

```java
public static ImagenPGM leerCabeceraPGM(String fileName) {
    FileInputStream archivoEntrada = null;
    Scanner sc = null;

    try {
        archivoEntrada = new FileInputStream(fileName);
        sc = new Scanner(archivoEntrada);
    } catch (FileNotFoundException e) {
        e.printStackTrace();
        return null;
    }

    String formato = sc.nextLine();
    int cantidadColumnas = sc.nextInt();
    int cantidadFilas = sc.nextInt();
    int maximo = sc.nextInt();
    sc.close();

    ImagenPGM imagen = new ImagenPGM(fileName,formato,cantidadColumnas,cantidadFilas,maximo);
    return imagen;
}
```

O(1)

La complejidad computacional de la función leerCabeceraPGM es **O(1)** ya que todas las sentencias corresponden a operaciones elementales y la sumatoria de n sentencias que tiene complejidad **O(1)** da como resultado **O(1)**.

```java
private static void leerPGM_P2(ImagenPGM imagen) {

    FileInputStream archivoEntrada = null;
    Scanner sc = null;

    try {
        archivoEntrada = new FileInputStream(imagen.getPath());
        sc = new Scanner(archivoEntrada);
    } catch (FileNotFoundException e) {
        e.printStackTrace();
        return ;
    }

    sc.nextLine();
    sc.nextLine();
    sc.nextLine();

    int minimo=imagen.getMaximoValor();
    for (int i = 0; i < imagen.getCantidadFilas(); i++) {
        for (int j = 0; j < imagen.getCantidadColumnas(); j++) {
            imagen.setImagen(i, j,  sc.nextInt());
            if(minimo>imagen.getImagen(i, j)) {
                minimo=imagen.getImagen(i, j);
            }
        }
    }
    imagen.setMinimo(minimo);
    sc.close();
}
```

O(1)

O(N*M)

O(1)

O(1)

La complejidad computacional de la función leerPGM_p2 es:

O(1) +O(N*M) + O(1) = O(N*M)

Nota: N corresponde a la cantidad de filas de la matriz y M corresponde a la cantidad de columnas

Análisis de La función leerImagenPGM:

```
33    public static void leerImagenPGM(ImagenPGM imagen){
34        if (imagen.getFormato().equals("P5")) {
35            leerPGM_P5(imagen);
36        } else if (imagen.getFormato().equals("P2")) {
37            leerPGM_P2(imagen);
38        }
39    }
```

T(n) = T(c) + max(T(leerPGM_p5,leerPGM_P2))
T(n)= k + T(N*M)

O(N*M)

Análisis de Complejidad de Función de Inversión de Píxeles:

```
58    public void invertirImagen() {
59        for (int i = 0; i < getCantidadFilas(); i++) {
60            for (int j = 0; j < getCantidadColumnas(); j++) {
61                setImagen(i, j,  maximoValor -getImagen(i,j));
62            }
63        }
64    }
65 }
```

O(N*M)

Análisis de Complejidad de Función de escritura de imágenes PGM:

```
18    private static void escribirPGM_P2(ImagenPGM imagen,String pathSalida) {
19
20        PrintWriter writer = null;
21        try {
22            writer = new PrintWriter(pathSalida);
23        } catch (FileNotFoundException e) {
24            e.printStackTrace();
25        }
26
27        writer.println("P2");
28        writer.println(imagen.getCantidadColumnas() + " " + imagen.getCantidadFilas());
29        writer.println(imagen.getMaximoValor()-imagen.getMinimo());
30
31        for (int i = 0; i < imagen.getCantidadFilas(); i++) {
32            for (int j = 0; j < imagen.getCantidadColumnas(); j++) {
33                writer.printf("%d" + " ",imagen.getImagen(i, j));  O(1)
34            }
35            writer.println();      O(1)
36        }
37
38        writer.close();
39    }
40
```

O(1)

O(N*M)

O(1)

La complejidad computacional de esta función es:

O(1) + O(N * M) + O(1)

Lo cual da como resultado una complejidad de **O(N*M)**.

```
10    public static void escribirImagenPGM(ImagenPGM imagen,String pathSalida){
11        if (imagen.getFormato().equals("P5")) {
12            escribirPGM_P5(imagen,pathSalida);
13        } else if (imagen.getFormato().equals("P2")) {
14            escribirPGM_P2(imagen,pathSalida);
15        }
16    }
```

T(N) = T(C) + MAX(T(escribirPGM_P5), T(escribirPGM_P2))
T(N)= K + T(N*M)
O(N*M)

Análisis del Algoritmo utilizando las funciones analizadas previamente

```
5    public static void main(String[] args) {
6
7        ImagenPGM imagen= LeerPGM.leerCabeceraPGM("Foto.pgm");      O(1)
8
9        LeerPGM.leerImagenPGM(imagen);                              O(N * M)
10
11       imagen.invertirImagen();                                    O(N * M)
12
13       EscribirPGM.escribirImagenPGM(imagen, "FotoSalida.pgm");    O(N * M)
14   }
15 }
```

Finalmente la complejidad computacional de este algoritmo de inversion de color es:

O(1) + O(N * M) + O(N * M) + O(N * M)

Lo cual da como resultado una complejidad de **O(N*M)**. En el caso de ser una matriz cuadrado la complejidad es de **O(N^2)**.

Aclaraciones: El ejemplo del cálculo se realizó utilizando las funciones correspondientes al procesado de imágenes PGM P2, para el caso de PGM P5 el análisis dio como resultado la misma complejidad computacional.

Análisis Empírico de la Complejidad Computacional del Algoritmo

Nombre Imagen	Filas	Columnas	Tiempo Ejecución (segundos)
Paisaje.pgm	4320	7680	203,6325887
Ola.pgm	1660	2560	26,1219963
Montaña.pgm	1024	1024	6,525794
gato.pgm	194	259	0,4571529
Foto.pgm	7	24	0,0985434

Procesamos distintas imágenes con diferentes dimensiones para ver como variaba el tiempo de ejecución respecto al tamaño de la entrada.

Conclusiones

El formato de imagen PGM (Portable Gray Map) es utilizado para almacenar imágenes en escala de grises, donde cada píxel es representado por un valor de brillo. Existen dos

variantes del formato: **P2 (ASCII)** y **P5 (Binario)** Ambos formatos comparten un encabezado en ASCII que posee la información de la imagen.

El algoritmo de inversion de Color consiste en modificar el valor de cada pixel de la imagen. Como vimos en el informe se puede realizar el cálculo mediante el complemento utilizando el valor máximo posible del archivo o el valor máximo que verdaderamente tiene el pixel.

El formato PGM permite modificar fácilmente imágenes en escalas grises. Provee una solución sencilla y útil para manejar este tipo de imágenes, permitiendo la aplicación de distintos filtros o transformaciones para poder alterarlas.

Bibliografía:

Adobe. (s.f.). *PGM Files*. Adobe. Recuperado el 3 de septiembre de 2024, de https://www.adobe.com/ar/creativecloud/file-types/image/raster/pgm-file.html
Wikipedia. (s.f.). *Formatos Netpbm (PGM)*. Recuperado el 3 de septiembre de 2024, de https://es.wikipedia.org/wiki/Formatos_Netpbm
YouTube. (2021, 6 de diciembre). *Invertir una imagen pgm | UPV* [Video]. Recuperado de https://www.youtube.com/watch?v=Da1t0YdV7q4
GeeksforGeeks. (2020, 16 de julio). *C program to invert (making negative) an image content in PGM format*. Recuperado de https://www.geeksforgeeks.org/c-program-to-invert-making-negative-an-image-content-in-pgm-format/
Colorado State University. (2018). *PictureLibrary.java*. Recuperado de https://www.cs.colostate.edu/~cs163/.Summer18/recitations/R12/files/PictureLibrary.java
GeeksforGeeks. (2019, 29 de junio). *Image processing in Java – Read and write*. Recuperado de https://www.geeksforgeeks.org/image-processing-in-java-read-and-write/