Universidad de Chile
Facultad de Ciencias Físicas y Matemáticas
Departamento de Ciencias de la Computación

Algoritmos y Estructuras de Datos

Tarea 2

Autor: Javier Gómez
Profesor: Benjamín Bustos
Auxiliares: Ignacio Valderrama
Manuel Olguín
Vanessa Peña Araya
Ayudantes: Fabián González
José Ignacio Moreno
Pablo Pizarro R.

Fecha de realización: 3 de octubre de 2016
Fecha de entrega: 4 de octubre de 2016
Santiago, Chile

Índice de Contenidos

1. Introducción

Los computadores para poder representar una imagen, toman un conjunto de pixeles a los cuales les asignan un color utilizando un modelo basado en la síntesis aditiva, denominado **RGB** (siglas que significan *Red Green Blue*), en donde se le asignan valores de los tres colores primarios a cada pixel y el color final resulta de la mezcla aditiva de estos.

Para una imagen, la cual estará representada con el formato ya explicado (pixeles con valores RGB), podemos trabajarla y transformarla. Un ejemplo de esto es marcar los bordes de una imagen. Para esto, se desarrollará un programa basado en Java, el cual recibirá los valores RGB para cada pixel de la imagen. Estos valores estarán representados en tres matrices, una para cada color (rojo, verde y azul), las cuales tendrán tamaño $m \times n$, donde m y n son el tamaño, en pixeles, de la imagen.

Estas matrices R,G y B serán operadas con otras que estarán determinadas por un *kernel* que será indicado por el usuario, para finalmente devolver una matriz con valores 0 y 1 (discriminados usando un valor de *gradiente*, tambien determinado por el usuario), la cual representará los pixeles de la imagen con los bordes marcados.

2. Diseño de la solución

La metodología que se utilizó para convertir una imagen (representada en matrices) a una que sólo contenga los bordes fue la siguiente:

El programa recibirá los valores de los colores de cada pixel y los guardará en tres matrices (R,G y B, las cuales corresponden a Rojo, Verde y Azul, respectivamente). Luego, con dichas matrices, usará la formula indicada en el enunciado de la tarea para calcular una matriz A que corresponderá a la imagen en tonos de grises. Una vez que haya creado esta matriz, calculará la convolución de A con dos matrices, M1 y M2, que dependerán del *kernel* que haya seleccionado el usuario. Luego de esto, usará las dos matrices resultantes para calcular el gradiente de A. Después, utilizando el valor *grad* ingresado previamente por el usuario, analizará cada valor de la matriz gradiente, y lo reemplazará por un 0 si es menor al valor de la variable *grad*, o por un 1 en el caso contrario. Luego de haber discriminado cada valor de la matriz, devolverá mediante la salida estándar la matriz resultante, que representará la imagen con los bordes marcados.

Para calcular la convolución, se utilizó un algoritmo iterativo, que va desplazando la matriz M sobre la matriz A, en donde el invariante cumple que *la matriz M nunca sale de la matriz A*, es decir, siempre está totalmente sobrepuesta en la matriz A, lo que se cumple siempre ya que se iteran las filas *m-p+1* veces (con $m = n°$ *de filas de A* y $p = n°$ *de filas de M*) y también se iteran las columnas *n-q+1* veces (con $n = n°$ *de columnas de A* y $q = n°$ *de columnas de M*), por lo que en cualquier ciclo de la iteración, los indices de M siempre estan contenidos en los de A.

El algoritmo dieseñado tomará tiempo de orden $O(m \cdot n)$, y al estar trabajando con matrices, este es un tiempo más que aceptable para lo requerido.

3. Implementación

A continuación, se revisará la implementación de la solución descrita en la sección anterior.

3.1. Recepción de valores y creación de Matrices R, G y B

En primer lugar se deben declarar las variables y matrices que utilizaremos, y leer los datos del usuario.

A continuación se muestra un extracto del código en donde se abre la clase ***main***, se inicializan las variables y matrices, y se leen los datos de la entrada estándar.

```
import java.io.IOException;
```

```
import java.util.Scanner;
import static java.lang.Math.*;

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

       (...) {
        String linea = in.nextLine();
        String[] param = linea.split(" ");
        m = Integer.parseInt(param[0]);
        n = Integer.parseInt(param[1]);
        grad = Integer.parseInt(param[2]);
        kernel = param[3];
        int[][] R = new int[m][n];
        int[][] G = new int[m][n];
        int[][] B = new int[m][n];

        for (int j = 0; j < m; ++j) {
            linea = in.nextLine();
            param = linea.split(" ");
            for (int h = 0; h < n; ++h) {
                R[j][h] = Integer.parseInt(param[h]);
            }
        }
        //hacemos el mismo ciclo para G y B (ver anexo)
        (...)
    }
```

Notemos que las matrices R, G y B son de tamaño $m \times n$, ya que la imagen es de $m \times n$ pixeles, por lo que cada elemento de la matriz representará a un pixel.

Utilizando la clase *Scanner*, leeremos los valores desde la entrada estándar.

3.2. Transformar las matrices R,G y B a la matriz A (tono de grises)

Una vez que tenemos creadas las matrices R, G y B, debemos transformarlas a una matriz A que representará la imagen en tono de grises. Para esto, usamos la fórmula entregada en el enunciado.

A continuación se muestra el código con esta fórmula implementada.

```
//inicializamos la matriz A
int[][] A = new int[m][n];
for (int i = 0; i < m; ++i) {
    for (int k = 0; k < n; ++k) {
        A[i][k] = (int) round(0.3 * R[i][k] + 0.59 * G[i][k] + 0.11 * B[i][k]);
    }
}
```

En este fragmento de código podemos ver que se usa una doble iteración para recorrer todos los espacios de la matriz A y de las matrices R, G y B

3.3. Calcular la convolución

El próximo paso es calcular la convolución de A con el *kernel* indicado por el usuario. Para esto, separaremos las instrucciones en tres casos según sea el nombre del *kernel* (se mostrará el caso del kernel *robert*. Para el código de los *kernel* restantes, ver anexo).

```
if (kernel.equals("robert")){
       //inicializamos las matrices M1 y M2 y le damos sus respectivos valores (omitidos en esta
           parte, para mas detalles ver anexo)

       //inicializamos las matrices que devolvera la convolucion
       int[][] C1 = new int[m-2+1][n-2+1];
       int[][] C2 = new int[m-2+1][n-2+1];
       int p = 2;
       int q = 2;
       for (int i = 0; i < m-p+1;++i){
           for (int j = 0; j < n-q+1;++j){
               C1[i][j] = A[i][j]*M1[0][0]+ A[i][j+1]*M1[0][1] + A[i+1][j]*M1[1][0] +
                   A[i+1][j+1]*M1[1][1];
               C2[i][j] = A[i][j]*M2[0][0]+ A[i][j+1]*M2[0][1] + A[i+1][j]*M2[1][0] +
                   A[i+1][j+1]*M2[1][1];
           }
       }
       //ya tenemos creadas C1 y C2
```

Cómo podemos ver, en los ciclos *for*, vamos moviendo las matrices M1 y M2 dentro de A y calculando la suma de la multiplicación de sus elementos. Es importante notar que M1 y M2 siempre están contenidas en A.

Esta operación nos devolverá dos matrices, C1 y C2, con las cuales podremos calcular el gradiente de A.

Es importante notar que las iteraciones llegan hasta *m-p+1* y *n-q+1*, ya que una matriz de $p \times q$ elementos, se puede sobreponer de *(m-p+1)* $\times$ *(n-q+1)* formas distintas sobre una matriz de $m \times n$, por lo que se generará una matriz de *(m-p+1)* $\times$ *(n-q+1)* elementos.

3.4. Calcular el gradiente de A

Una vez que ya tenemos creadas nuestras matrices C1 y C2, provenientes de la convolución, podemos calcular la matriz gradiente con la matriz que se nos entrega en el enunciado.

```
double[][] Gr = new double[m-2+1][n-2+1];
for (int i = 0; i < m-p+1;++i) {
    for (int j = 0; j < n - q + 1; ++j) {
        Gr[i][j] = Math.sqrt((C1[i][j]*C1[i][j] + C2[i][j]*C2[i][j]));

    }
}
```

Para cada elemento de las matrices C1 y C2 (cabe destacar que son del mismo tamaño), calculamos: $Gr(i,j) = \sqrt{C1(i,j)^2 + C2(i,j)^2}$, y con esto poder obtener nuestra matriz gradiente.

3.5. Cálculo de los bordes

Para calcular los bordes, primero inicializaremos una matriz *borde*, que contendrá valores 0 o 1, dependiendo del valor de la variable *grad* que haya indicado el usuario.

```
float[][] borde = new float[m-2+1][n-2+1];
for (int i = 0; i < m-p+1;++i) {
    for (int j = 0; j < n - q + 1; ++j) {
        if (Gr[i][j] < grad){
            borde[i][j] = 0;}
        else { borde[i][j] = 1;}
```

```
    }
}
```

Se recorrerá la matriz *Gr* por cada uno de sus elementos. Cuando estos sean menores a *grad*, se colocará un 0 en la matriz *borde* (en la misma posicion del elemento de *Gr*), y en el caso contrario (mayores o iguales a *grad*), se colocará un 1.

3.6. Impresión de matriz *borde* en la salida estándar

Una vez calculada la matriz *borde*, sólo falta imprimir la matriz resultante mediante la salida estándar.

```
for (int i = 0; i < m-p+1;++i) {
    for (int j = 0; j < n - q; ++j) {
        System.out.print(borde[i][j] + " ");

    }
    System.out.print(borde[i][n - q]);
    //hacemos un salto de linea
    System.out.println();
    }
}
```

Con esto, tendremos en la salida estándar nuestra matriz, que indicará los bordes de la imagen.

4. Resultados y Conclusiones

El programa recién analizado, nos permite marcar los bordes de una imagen según los criterios que deseemos. Esta herramienta puede ser útil para manejar y/o retocar imágenes al gusto que el usuario necesite.

Para cualquier tamaño de matriz que ingresemos, obtendremos una nueva matriz de bordes, que nos permitirá generar la nueva imagen mediante el uso de una herramienta que nos permita esto.

Como podemos ver, no es difícil implementar un algoritmo que trabaje matrices que le sean entregadas, siempre y cuando sepamos realizar satisfactoriamente las operaciones que queramos hacer (en este caso, calcular el gradiente y la convolución mediante las fórmulas que se nos fueron entregadas).

Cabe destacar que el algoritmo utilizado toma tiempo de orden $O(m \cdot n)$, aceptable para lo requerido. Este dato podría ser importante a la hora de analizar la implementación del programa o de desarrollar un algoritmo distinto a partir del ya realizado.

5. Anexo 1: Código del programa

```
import java.io.IOException;
import java.util.Scanner;
import static java.lang.Math.*;

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

        int m;
        int n;
        int grad;
        String kernel;
        Scanner in = new Scanner(System.in);
        while (in.hasNextLine()) {

        String linea = in.nextLine();
        String[] param = linea.split(" ");
        m = Integer.parseInt(param[0]);
        n = Integer.parseInt(param[1]);
        grad = Integer.parseInt(param[2]);
        kernel = param[3];
        int[][] R = new int[m][n];
        int[][] G = new int[m][n];
        int[][] B = new int[m][n];

        for (int j = 0; j < m; ++j) {
            linea = in.nextLine();
            param = linea.split(" ");
            for (int h = 0; h < n; ++h) {
                R[j][h] = Integer.parseInt(param[h]);}}

        for (int j = 0; j < m; ++j) {
            linea = in.nextLine();
            param = linea.split(" ");
            for (int h = 0; h < n; ++h) {
                G[j][h] = Integer.parseInt(param[h]);}}

        for (int j = 0; j < m; ++j) {
            linea = in.nextLine();
            param = linea.split(" ");
            for (int h = 0; h < n; ++h) {
                B[j][h] = Integer.parseInt(param[h]);}}

        int[][] A = new int[m][n];
        for (int i = 0; i < m; ++i) {
            for (int k = 0; k < n; ++k) {
                A[i][k] = (int) round(0.3 * R[i][k] + 0.59 * G[i][k] + 0.11 * B[i][k]);}}

        if (kernel.equals("robert")){
        int[][] M1 = new int[2][2];
        int[][] M2 = new int[2][2];
        M1[0][0] = 0;
        M1[1][0] = -1;
        M1[0][1] = 1;
```

```
M1[1][1] = 0;

M2[0][0] = 1;
M2[1][0] = 0;
M2[0][1] = 0;
M2[1][1] = -1;
int[][] C1 = new int[m-2+1][n-2+1];
int[][] C2 = new int[m-2+1][n-2+1];
int p = 2;
int q = 2;

for (int i = 0; i < m-p+1;++i){
    for (int j = 0; j < n-q+1;++j){
        C1[i][j] = A[i][j]*M1[0][0]+ A[i][j+1]*M1[0][1] + A[i+1][j]*M1[1][0] +
            A[i+1][j+1]*M1[1][1];
        C2[i][j] = A[i][j]*M2[0][0]+ A[i][j+1]*M2[0][1] + A[i+1][j]*M2[1][0] +
            A[i+1][j+1]*M2[1][1];}}
//ya tenemos creadas C1 y C2
double[][] Gr = new double[m-2+1][n-2+1];
for (int i = 0; i < m-p+1;++i) {
    for (int j = 0; j < n - q + 1; ++j) {
        Gr[i][j] = Math.sqrt((C1[i][j]*C1[i][j] + C2[i][j]*C2[i][j]));}}
float[][] borde = new float[m-2+1][n-2+1];
for (int i = 0; i < m-p+1;++i) {
    for (int j = 0; j < n - q + 1; ++j) {
        if (Gr[i][j] < grad){
            borde[i][j] = 0;}
        else { borde[i][j] = 1;}}}
//RETORNAR AQUI
for (int i = 0; i < m-p+1;++i) {
    for (int j = 0; j < n - q; ++j) {
        System.out.print(borde[i][j] + " ");}
    System.out.print(borde[i][n - q]);
    //hacemos un salto de linea
    System.out.println();}}

if (kernel.equals("prewitt")) {
    int[][] M1 = new int[3][3];
    int[][] M2 = new int[3][3];
    M1[0][0] = -1;
    M1[1][0] = -1;
    M1[0][1] = 0;
    M1[1][1] = 0;
    M1[0][2] = 1;
    M1[1][2] = 1;
    M1[2][0] = -1;
    M1[2][1] = 0;
    M1[2][2] = 1;

    M2[0][0] = -1;
    M2[1][0] = 0;
    M2[0][1] = -1;
    M2[1][1] = 0;
    M2[0][2] = -1;
    M2[1][2] = 0;
    M2[2][0] = 1;
```

```
        M2[2][1] = 1;
        M2[2][2] = 1;

        int[][] C1 = new int[m - 3 + 1][n - 3 + 1];
        int[][] C2 = new int[m - 3 + 1][n - 3 + 1];
        int p = 3;
        int q = 3;

        for (int i = 0; i < m - p + 1; ++i) {
            for (int j = 0; j < n - q + 1; ++j) {
                C1[i][j] = A[i][j] * M1[0][0] + A[i][j + 1] * M1[0][1] + A[i + 1][j] * M1[1][0] +
                    A[i + 1][j + 1] * M1[1][1] + A[i + 2][j] * M1[2][0] + A[i][j + 2] * M1[0][2]
                    + A[i + 1][j + 2] * M1[1][2] + A[i + 2][j + 1] * M1[2][1] + A[i + 2][j + 2] *
                    M1[2][2];
                C2[i][j] = A[i][j] * M2[0][0] + A[i][j + 1] * M2[0][1] + A[i + 1][j] * M2[1][0] +
                    A[i + 1][j + 1] * M2[1][1] + A[i + 2][j] * M2[2][0] + A[i][j + 2] * M2[0][2]
                    + A[i + 1][j + 2] * M2[1][2] + A[i + 2][j + 1] * M2[2][1] + A[i + 2][j + 2] *
                    M2[2][2];}}
        //ya tenemos creadas C1 y C2
        double[][] Gr = new double[m - 2 + 1][n - 2 + 1];
        for (int i = 0; i < m - p + 1; ++i) {
            for (int j = 0; j < n - q + 1; ++j) {
                Gr[i][j] = Math.sqrt((C1[i][j] * C1[i][j] + C2[i][j] * C2[i][j]));}}
        float[][] borde = new float[m - 2 + 1][n - 2 + 1];
        for (int i = 0; i < m - p + 1; ++i) {
            for (int j = 0; j < n - q + 1; ++j) {
                if (Gr[i][j] < grad) {
                    borde[i][j] = 0;
                } else {
                    borde[i][j] = 1;}}}
        //RETORNAR AQUI
        for (int i = 0; i < m-p+1;++i) {
            for (int j = 0; j < n - q; ++j) {
                System.out.print(borde[i][j] + " ");
            }
            System.out.print(borde[i][n - q]);
            //hacemos un salto de linea
            System.out.println();}}

    if (kernel.equals("sobel")){
        int[][] M1 = new int[3][3];
        int[][] M2 = new int[3][3];
        M1[0][0] = -1;
        M1[1][0] = -2;
        M1[0][1] = 0;
        M1[1][1] = 0;
        M1[0][2] = 1;
        M1[1][2] = 2;
        M1[2][0] = -1;
        M1[2][1] = 0;
        M1[2][2] = 1;

        M2[0][0] = -1;
        M2[1][0] = 0;
        M2[0][1] = -2;
        M2[1][1] = 0;
```

```
            M2[0][2] = -1;
            M2[1][2] = 0;
            M2[2][0] = 1;
            M2[2][1] = 2;
            M2[2][2] = 1;

            int[][] C1 = new int[m-3+1][n-3+1];
            int[][] C2 = new int[m-3+1][n-3+1];
            int p = 3;
            int q = 3;

            for (int i = 0; i < m-p+1;++i){
                for (int j = 0; j < n-q+1;++j){
                    C1[i][j] = A[i][j]*M1[0][0]+ A[i][j+1]*M1[0][1] + A[i+1][j]*M1[1][0] +
                        A[i+1][j+1]*M1[1][1] + A[i+2][j]*M1[2][0] + A[i][j+2]*M1[0][2] +
                        A[i+1][j+2]*M1[1][2] + A[i+2][j+1]*M1[2][1] + A[i+2][j+2]*M1[2][2];
                    C2[i][j] = A[i][j]*M2[0][0]+ A[i][j+1]*M2[0][1] + A[i+1][j]*M2[1][0] +
                        A[i+1][j+1]*M2[1][1] + A[i+2][j]*M2[2][0] + A[i][j+2]*M2[0][2] +
                        A[i+1][j+2]*M2[1][2] + A[i+2][j+1]*M2[2][1] + A[i+2][j+2]*M2[2][2];}}
            //ya tenemos creadas C1 y C2
            double[][] Gr = new double[m-2+1][n-2+1];
            for (int i = 0; i < m-p+1;++i) {
                for (int j = 0; j < n - q + 1; ++j) {
                    Gr[i][j] = Math.sqrt((C1[i][j]*C1[i][j] + C2[i][j]*C2[i][j]));}}
            float[][] borde = new float[m-2+1][n-2+1];
            for (int i = 0; i < m-p+1;++i) {
                for (int j = 0; j < n - q + 1; ++j) {
                    if (Gr[i][j] < grad){
                        borde[i][j] = 0;
                    }
                    else { borde[i][j] = 1;}}}
            //RETORNAR AQUI AQUI
            for (int i = 0; i < m-p+1;++i) {
                for (int j = 0; j < n - q; ++j) {
                    System.out.print(borde[i][j] + " ");}
                System.out.print(borde[i][n - q]);
                //hacemos un salto de linea
                System.out.println();}
        }
        }
    }
}
```