



Laboratorio de Estructuras de Datos

Práctica 9. Arreglos multidimensionales

Unidad Temática: 1. Introducción a las estructuras de datos y estructuras fundamentales

📅 👤 Profesor: Dr. Aldonso Becerra Sánchez

Índice

1	Objetivo de la tarea	1
2	Tiempo aproximado de realización	1
3	Fecha de entrega	1
4	Fecha de entrega con extensión y penalización	1
5	Introducción	1
6	Actividades a realizar	1
6.1	Actividad Inicial	1
6.2	Actividad 1	1
6.3	Actividad 2	1
6.4	Actividad 3	4
6.5	Actividad 4	4
6.6	Actividad 5	4
6.7	Actividad 6	4
6.8	Actividad 7	4
7	Contáctame	4
	References	4

1. Objetivo de la tarea

Comprender el uso de los arreglos multidimensionales para la resolución de un problema real.

2. Tiempo aproximado de realización

🕒 5 horas.

3. Fecha de entrega

📅 22 de septiembre de 2024.

4. Fecha de entrega con extensión y penalización

📅 23 de septiembre agosto de 2024.

5. Introducción

La facilidad que los arreglos multidimensionales tienen para permitir guardar datos en forma de columnas/renglones, los hace pertinentes para la resolución de muchos problemas donde se requiere esta situación. El único detalle con esta cuestión es que es poco flexible el número de elementos que podemos manipular, ya

que se requiere conocer a priori la cantidad de elementos a guardar. Adicionalmente se incrustan los usos de registros para darle una funcionalidad más completa al procedimiento de control [1], [2], [3].

6. Actividades a realizar

6.1. Actividad Inicial

Lea primero toda la práctica ⚠️. No inicie a programar sin leer todo cuidadosamente primero. Recuerde que debe generar el reporte en formato IDC con todos sus componentes.

6.2. Actividad 1

Primero genere la Introducción 📄.

6.3. Actividad 2

Información importante

Esta actividad debe entrar en la parte de *Desarrollo* 📄.

Las propiedades del color pueden ser definidas matemáticamente usando modelos de color, los cuales describen los colores que vemos y con los que trabajamos. Cada modelo de color representa un método diferente de descripción y clasificación del color, pero todos utilizan valores numéricos para representar el espectro visible. Existen muchos modelos, uno de los principales es RGB (canales red, green, blue). Si ha trabajado con un programa de edición de imágenes, quizás les resulte familiar otros modelos como Escala de grises. En RGB, además de los tres colores base se puede tener el canal alfa. En computación gráfica, la composición alfa o canal alfa es la que define la opacidad de un píxel en una imagen. El canal alfa actúa como una máscara de transparencia que permite, de forma virtual, componer (mezclar capas) imágenes o fondos opacos con imágenes con un cierto grado de transparencia. Para indicar con qué proporción es mezclado cada color, se asigna un valor a cada uno de los colores primarios, de manera que el valor "0" significa que no interviene en la mezcla y, a medida que ese valor aumenta, se entiende que aporta más intensidad a la mezcla. Aunque el intervalo de valores podría ser cualquiera (valores reales entre 0 y 1, valores enteros entre 0 y 37, etc.), es frecuente que cada color primario se codifique con un byte (8 bits). Así, de manera usual, la intensidad de cada una de las componentes se mide según una escala que va del 0 al 255 y cada color es definido por un conjunto de valores escritos entre paréntesis (correspondientes a valores R", "Gz "B", y en su caso alfa) y separados por comas. De este modo, el rojo se obtiene con (255, 0, 0), el verde con (0, 255, 0) y el azul con (0, 0, 255), obteniendo, en cada caso un color resultante monocromático.

La ausencia de color, es decir el color negro, se obtiene cuando las tres componentes son 0: (0, 0, 0). La combinación de dos colores a su máximo valor de 255 con un tercero con valor 0 da lugar a tres colores intermedios. De esta forma, aparecen los colores amarillo (255, 255, 0), cian (0, 255, 255) y magenta (255, 0, 255). El color blanco se forma con los tres colores primarios a su máximo valor (255, 255, 255). Adicionalmente a estos valores, se puede obtener el canal alfa, como cuarto elemento. Bajo este supuesto una imagen de 800x600 está compuesta por 800 pixeles de ancho por 600 pixeles de alto. Donde cada pixel es representado por un byte por cada canal, si se tiene 3 canales (RGB) se disponen de 24 bits, si se tienen 4 canales (alfa y RGB) se disponen de 32 bits en la configuración de un pixel. En otras palabras un pixel se puede ver así: La PROFUNDIDAD DE BITS es determinada por la cantidad de bits utilizados para definir cada pixel. Cuanto mayor sea la profundidad de bits, tanto mayor será la cantidad de tonos (escala de grises o color) que puedan ser representados. Las imágenes digitales se pueden producir en blanco y negro (en forma bitonal), a escala de grises o a color. ¿Cuántos colores puede tener un pixel? De todos es sabido que los archivos y dispositivos digitales almacenan números, concretamente números en sistema binario, es decir ceros y unos, nada más. El color que queremos mostrar en un determinado pixel lo representamos pues con un número. ¿cuántos colores podremos mostrar? Depende de la cantidad de información asociada con cada pixel.

1 bit: 0 1. El ejemplo más sencillo sería este: utilizar 1 bit para la información de color. Un bit puede valer 0 o 1. Por tanto sólo podremos representar 2 tonos. Suele utilizarse 0 para el blanco y 1 para el negro.

2 bits: 00 10 01 11, con dos bits podemos representar 4 tonos diferentes.

3 bits: 000 001 010 100 110 101 011 111, es decir 8 colores diferentes.

4 bits: 16 colores (ver Figura 1).



Figura 1. Ejemplo de imagen de 16 colores.

8 bits: 256 colores. Esta es la profundidad de color habitual en las imágenes de “escala de grises”: los píxeles pueden tomar uno entre 256 tonos de gris (ver Figura 2).

24 bits: 16,7 millones de colores. El estándar para imágenes en color es una profundidad de 24 bits, dedicándose 8 bits a almacenar la información de cada uno de los 3 colores primarios: 8 bits para el Rojo, 8 para el Verde y 8 para el Azul. Es decir, una imagen estándar en color puede almacenar 256 tonos de rojo, 256 de verde y 256 de azul: $256 \times 256 \times 256 = 16.777.216$ colores. A las imágenes que permiten esta variedad tonal (24 bits, o 8 bits por canal) se las denomina imágenes en color verdadero o más habitualmente con su equivalente inglés “true color” (ver Figura 3).

Partiendo de un tipo de dato (en un lenguaje de programación) que permita guardar 32 bits (int, por ejemplo), los primeros 8 bits del



Figura 2. Ejemplo de imagen de 256 colores.

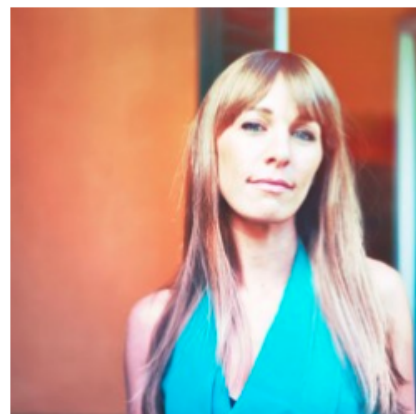


Figura 3. Ejemplo de imagen de 16.7 millones de colores.

lado izquierdo representan al canal alfa (opacidad), los siguientes 8 bits representan el color rojo, los siguientes 8 bits representan el color verde, los siguientes 8 bits (lo más a la derecha) representan el color azul.

Alfa	Red	Green	Blue
8 bits	8 bits	8 bits	8 bits
01010101	00001111	01011100	00001110

Se le pide que realice lo siguiente:

- Se proporciona un código de programa (en la parte inferior de este documento) que lee una imagen indicada como ruta. El método `imagen.getRGB(i, j)` obtiene un número de 32 bits (un pixel) representando los 4 canales mencionados (alfa, r, g, b). Se le pide que guarde cada pixel de la imagen en un TDA, donde las dimensiones requeridas son posible obtenerlas con base en las dimensiones de la imagen (`imagen.getWidth()` y `imagen.getHeight()`).
- Extraiga de cada pixel (número de 32 bits) los correspondientes valores de cada canal en una variable separada para cada uno. Utilice operaciones a nivel de bits, como corrimientos hacia la derecha o izquierda para manipular los bits que se requieran (inclusive se pueden utilizar AND o OR lógicos según sea el caso). Utilice su raciocinio para deducir estos corrimientos con base en los 32 bits que vienen indicados más arriba. Recuerde que un número entero en java es de 32 bits, y que un pixel utiliza los 32 bits para guardar su valor. De esta manera se requiere por ejemplo, para extraer el Green, que sus 8 bits se localicen hasta la derecha de los 32 bits (se recorrieron 8 bits hacia la derecha), y que todos los demás bits sean ceros (los 24 de la izquierda). Así se tiene guardado el color Green en los primeros 8 bits una vez que se ha movido el conjunto de bits. Este valor está guardado

en un entero de 32 bits, pero representando solo los bits de ese color (Green, por ejemplo). Este proceso lo tiene que hacer para extraer los demás colores por individual.

3. Tome como base una imagen jpg de su elección, y pase la imagen a escala de grises. Este proceso se obtiene promediando los valores individuales de cada canal. El resultado del promedio será el nuevo valor que tendrá cada canal (el mismo valor para cada canal). Con esto valores se debe formar el nuevo valor del pixel de 32 bits (con la posición de bits que le corresponde a cada canal). Este proceso se obtiene nuevamente usando operaciones a nivel de bits. El proceso siguiente es pasar cada pixel a la imagen que se va a grabar con el método `imagen.setRGB(i, j, valorPixel32bits)`. El resultado de esta imagen ya se puede guardar en un archivo nuevo (ver Figuras 4 y 5).

NOTA: la siguiente línea crea una nueva imagen en blanco de tamaño w, h: `BufferedImage imagen2 = new BufferedImage(w, h, BufferedImage.TYPE_INT_RGB)`.

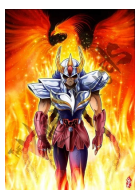


Figura 4. Imagen normal.



Figura 5. Imagen escala de grises.

4. Ahora se le pide que modifique el brillo de la imagen (en la imagen de escala de grises). Este proceso se obtiene sumando o restando un valor numérico a cada pixel por igual (ver Figuras 6, 7, 8, 9 y 10).

Figura 6.
Imagen a
color.Figura 7.
Imagen escala
de grises.Figura 8.
Imagen escala
de grises.Figura 9.
Imagen escala
de grises.Figura 10.
Imagen escala
de grises.

5. Tome como base la misma imagen y ahora invierta la imagen en posición horizontal (ver Figuras 11 y 12).
6. Tome como base la misma imagen y ahora invierta la imagen en posición vertical (ver Figuras 13 y 14).
7. Tome como base la misma imagen y ahora gire la imagen en 90, 180, 270 grados (ver Figuras 15 y 16, 17 y 18, 19 y 20).
8. Si utiliza las operaciones que se tienen en el TDA matriz, y ejecuta la operación transpuesta sobre la imagen usada de ejemplo, qué obtendría. Muestre el resultado y ejecute ese proceso.
9. Redimensione el tamaño de una imagen proporcionada como base (deben ser dos métodos sobrecargados, uno que indique el



Figura 11. Imagen normal.



Figura 12. Imagen girada.



Figura 13. Imagen normal.



Figura 14. Imagen girada.

doble, el triple, el cuádruple, la mitad la cuarta parte etc.; y el otro que permita hacerlo con base en tamaño pero en pixeles). Por ejemplo, se puede indicar, si se tiene una imagen de 400x500, que se aumente el tamaño al doble: 800x1000. Que se aumente al triple, etc. O se le puede pedir que disminuya su tamaño, a la mitad, a la tercera parte, etc. O por ejemplo se le pida que la imagen quede de 500x500, 800x1000, etc. Este argumento debe ser pasado al usuario a través de un enumerado nuevo que usted defina, para el primer método o en unas variables de ancho y alto para el segundo método. Ejemplo, una imagen aumentada al doble de tamaño sería lo que se muestra en las Figuras 21 y 22. Y una imagen disminuida a la mitad de tamaño sería lo que se muestra en las Figuras 23 y 24.

10. Se le pide que le haga un marco a la imagen sin robar espacio de la propia imagen (el usuario debe elegir un color y el grosor).
11. Se le pide que invente un nuevo método a partir de los ejemplos de arriba, de tal manera que usted indique qué operación haría y cómo quedaría el resultado. Por obvias razones debe ser un método nuevo que manipule la imagen de alguna manera diferente a las ya mostradas arriba.

Haga el programa (actividad 2, la cual es el Desarrollo, junto con la captura de pantalla del programa funcionando). Use una imagen que usted desee de manera personalizada.

Por ejemplo, para manipular los pixeles de imagen en lectura y escritura:

```
1
2 import javax.imageio.ImageIO;
3 import java.awt.*;
```



Figura 15. Imagen normal.



Figura 16. Imagen girada.



Figura 19. Imagen normal.



Figura 20. Imagen girada.



Figura 17. Imagen normal.



Figura 18. Imagen girada.



Figura 21. Imagen normal.



Figura 22. Imagen modificada.

```

4 import java.awt.image.BufferedImage;
5 import java.io.ByteArrayOutputStream;
6 import java.io.File;
7 import java.io.FileInputStream;
8 import java.io.IOException;
9
10 //codigo .....
11
12 BufferedImage imagen = ImageIO.read(new File("
    src/avefenix.jpg"));
13
14     int w = imagen.getWidth();
15     int h = imagen.getHeight();
16     BufferedImage imagen2 = new
    BufferedImage(w, h, BufferedImage.
    TYPE_INT_RGB);
17
18     //codigo .....
19
20     int pixel = imagen.getRGB(i, j);
21
22     //codigo ....
23     imagen2.setRGB(i, j, colorInt);
24
25     //codigo ...
26     ImageIO.write(imagen2, "JPG", new File("
    src/espejo.jpg"));
27
28     //codigo ...

```


6.4. Actividad 3

Pruebe el funcionamiento del programa de la actividad 2 con todo y sus capturas de pantalla.


6.5. Actividad 4

Realice la sección de Código agregado  (diagrama de clases UML).


6.6. Actividad 5

Realice la sección de Pre-evaluación  (use los lineamientos establecidos).

6.7. Actividad 6

Finalmente haga las Conclusiones .


6.8. Actividad 7


Subir los entregables (pdf  y zip  con código )  a Moodle.


7. Contáctame

Puedes contactarme a través de los siguientes medios.

 <https://moodle.ingsoftware.uaz.edu.mx/>

 a7donso@gmail.com

 Cubículo

 Salón CC2-IS

Referencias

- [1] O. Cairo y S. Guardati, *Estructura de datos*. McGraw-Hill.
- [2] L. Joyanes Aguilar, *Fundamentos de programación, algoritmos u estructura de datos*. McGraw-Hill.



Figura 23. Imagen normal.



Figura 24. Imagen modificada.

[3] M. A. Weiss, *Estructura de datos en Java*. Addison Wesley.