EJERCICIO 5 PRÁCTICA IMAGEN

1.Comprobar que efectivamente funciona ShuffleRows().

En primer momento debemos de comprobar que efectivamente Image Image::ShuffleRows() funciona correctamente y efectivamente tras crear barajar.cpp y ejecutar el ejemplo que se exige, funciona. Nos queda así:

PEGAR DESDE WINDOWS IMÁGENES DE ANTES Y DESPUÉS

2.Eficiencia empírica de Image Image::ShuffleRows() propuesto en el pdf con la implementación de la matriz que figura en la página 8 del pdf.

Para ello vamos a medir el tiempo que tarda variando el número de filas, de columnas y veces en las que se llama a la función ShuffleRows(), sirviéndonos de la librería <ctime> que nos va a ser de gran ayuda para desempeñar la tarea.

El código sería este:

```
// Fichero: barajar.cpp
// Baraja las filas de una imagen PGM
//

#include <iostream>
#include <cstring>
#include <cstdlib>
#include<ctime>

#include <image.h>

using namespace std;

int main (int argc, char *argv[]){
  char *origen, *destino; // nombres de los ficheros
 Image image;

 //vemos el tiempo inicial
 clock_t tini;
 tini=clock();

 // Comprobar validez de la llamada

 // Obtener argumentos
 origen  = argv[1];
 destino = argv[2];
```

```
 // Mostramos argumentos
 cout << endl;
 cout << "Fichero origen: " << origen << endl;
 cout << "Fichero resultado: " << destino << endl;

 // Leer la imagen del fichero de entrada
 if (!image.Load(origen)){
   cerr << "Error: No pudo leerse la imagen." << endl;
   cerr << "Terminando la ejecucion del programa." << endl;
   return 1;
 }

 // Mostrar los parametros de la Imagen
 cout << endl;
 cout << "Dimensiones de " << origen << ":" << endl;
 cout << "   Imagen    = " << image.get_rows()  << " filas x " <<
image.get_cols() << " columnas " << endl;

 // Baraja las filas de una imagen
 image.ShuffleRows();
  // Guardar la imagen resultado en el fichero
 if (image.Save(destino))
   cout  << "La imagen se guardo en " << destino << endl;
 else{
   cerr << "Error: No pudo guardarse la imagen." << endl;
   cerr << "Terminando la ejecucion del programa." << endl;
   return 1;
 }

 //una vez ha realizado las tareas medimos de nuevo el tiempo

 clock_t tfin;
 tfin=clock();

 // mostramos los resultados

 cout << "Ticks de reloj : " << tfin-tini << endl;
 cout << "Segundos : " << (tfin-tini)/(double)CLOCKS_PER_SEC << endl;


 return 0;
}
```

Y la salida que tenemos con la imagen que se nos propone es:

ismael@ismael-Spin-SP313-51N:~/Practicas_Estructura_De_Datos/Practica1/elbueno/ej01_imagenes$ ./barajar img/shuffle_9973.pgm output/baraja.pgm

Fichero origen: img/shuffle_9973.pgm
Fichero resultado: output/baraja.pgm

Dimensiones de img/shuffle_9973.pgm:
  Imagen = 256 filas x 256 columnas
La imagen se guardo en output/baraja.pgm
Ticks de reloj : 2465
Segundos : 0.002465

Esta salida era para una foto de formato 256*256 píxeles.

Usamos la función zoom para aumentar de tamaño la imagen, de esta manera generamos una imagen más grande para poder usarla como ejemplo y calcular el tiempo que tarda.

./zoom img/shuffle_9973.pgm output/zoom_shuffle.pgm 0 0 255

Fichero origen: img/shuffle_9973.pgm
Fichero resultado: output/zoom_shuffle.pgm

Dimensiones de img/shuffle_9973.pgm:
  Imagen = 256 filas x 256 columnas
La imagen se guardo en output/zoom_shuffle.pgm

Nos queda una imagen de 509 píxeles.

Y como vemos nos da esta salida:
ismael@ismael-Spin-SP313-51N:~/Practicas_Estructura_De_Datos/Practica1/elbueno/ej01_imagenes$ ./barajar output/zoom_shuffle.pgm output/baraja_509pixeles.pgm

Fichero origen: output/zoom_shuffle.pgm
Fichero resultado: output/baraja_509pixeles.pgm

Dimensiones de output/zoom_shuffle.pgm:
  Imagen = 509 filas x 509 columnas
La imagen se guardo en output/baraja_509pixeles.pgm
Ticks de reloj : 4615
Segundos : 0.004615

Obviamente tarda más tiempo.

Y ahora vamos a probar con una imagen de menos píxeles.

ismael@ismael-Spin-SP313-51N:~/Practicas_Estructura_De_Datos/Practica1/elbueno/ej01_imagenes$ ./zoom img/shuffle_9973.pgm output/zoom_shuffle.pgm 0 0 50

Fichero origen: img/shuffle_9973.pgm
Fichero resultado: output/zoom_shuffle.pgm

Dimensiones de img/shuffle_9973.pgm:
  Imagen = 256 filas x 256 columnas
La imagen se guardo en output/zoom_shuffle.pgm

Nos queda una imagen de 99*99 píxeles, que tarda:

ismael@ismael-Spin-SP313-51N:~/Practicas_Estructura_De_Datos/Practica1/elbueno/ej01_imagenes$ ./barajar output/zoom_shuffle.pgm output/baraja_99pixeles.pgm

Fichero origen: output/zoom_shuffle.pgm
Fichero resultado: output/baraja_99pixeles.pgm

Dimensiones de output/zoom_shuffle.pgm:
  Imagen = 99 filas x 99 columnas
La imagen se guardo en output/baraja_99pixeles.pgm
Ticks de reloj : 361
Segundos : 0.000361

Y obviamente tarda menos que la original.

Ahora vamos añadir una función ShuffleRows() más, además de un load() y save () adicional, y vemos como afecta a la complejidad experimental.

Básicamente hemos duplicado dichas funciones en el código.
Nos da como salida:
./barajar img/shuffle_9973.pgm output/baraja.pgm

Fichero origen: img/shuffle_9973.pgm
Fichero resultado: output/baraja.pgm

Dimensiones de img/shuffle_9973.pgm:
  Imagen = 256 filas x 256 columnas
La imagen se guardo en output/baraja.pgm
La imagen se guardo en output/baraja.pgm
Ticks de reloj : 5060
Segundos : 0.00506

Y podemos observar que la complejidad experimental aumenta.

3.Cambio de la implementación interna de la imagen

Ahora vamos a cambiar la implementación interna, básicamente como se inicializa la matriz.

Pasamos de esto:

```
void Image::Allocate(int nrows, int ncols, byte * buffer){
   rows = nrows;
   cols = ncols;

   img = new byte * [rows];

   if (buffer != 0)
       img[0] = buffer;
   else
       img[0] = new byte [rows * cols];

   for (int i=1; i < rows; i++)
       img[i] = img[i-1] + cols;
}
```

A esto:

```
void Image::Allocate(int nrows, int ncols, byte* buffer) {
   rows = nrows;
   cols = ncols;
   img = new byte*[rows];
   //inicializa la matriz con filas no contiguas en memoria
   for (int i = 0; i < rows; ++i) {
           img[i] = new byte[cols];
       }

   if (buffer != nullptr) {
       // Utiliza el búfer proporcionado
       int pos=0;
       for (int i = 0; i < rows; ++i) {
           for(int j=0;j<cols;++j){
               img[i][j]=buffer[pos++];
           }
       }
   }
}
```

Y vemos que he optado por el cambio de img[i]=new byte [cols].
Vamos a comprobar que las demás funciones funcionan correctamente con este cambio.

Y en mi caso las demás funciones funcionan igual.A excepción de :

-Debemos de cambiar el copy, ya que esta establecido para recorrer rows*cols, debemos cambiarlo por dos for anidados.
-El Destroy ya que tenemos una implementación de matriz distinta.
-Los set y get pixel de la imagen desenrollada. En este punto tenemos dos opciones o bien cambiar el constructor, o mantenerlo pero cambiar dichas funciones de get y set desenrollado. En mi caso optamos por la segunda mencionada, y pasamos de image [0][k] a img[i][j], siendo i =k/rows (para localizar la fila) y k%cols (para localizar la columna).
-El método save: pasamos de devolver copiar unicamente img[0] a tener que crear un array auxiliar copiarlo de manera que nos sea útil y devolverlo, esta explicado en Image.cpp.

El pdf expone que puede variar save, pero en mi caso ejecutando:

./barajar img/shuffle_9973.pgm output/baraja.pgm

Me da la misma salida que con la implementación anterior.

4.Ajustar la nueva implementación de la matriz a la función ShuffleRows()

Sin cambiar la implementación interna de la matriz nos da esta salida:

ismael@ismael-Spin-SP313-51N:~/Practicas_Estructura_De_Datos/Practica1/elbueno/ej01_imagenes$ ./barajar output/zoom_shuffle.pgm output/baraja_99pixeles.pgm

Fichero origen: output/zoom_shuffle.pgm
Fichero resultado: output/baraja_99pixeles.pgm

Dimensiones de output/zoom_shuffle.pgm:
  Imagen = 99 filas x 99 columnas
La imagen se guardo en output/baraja_99pixeles.pgm
Ticks de reloj : 436
Segundos : 0.000436

Donde podemos ver que el tiempo disminuye respecto de la anterior implementación: pasa de 361 a 436, es decir, empeora, ¿hemos hecho lo correcto? Debemos de ajustar la implementación del ShuffleRows para aprovecharla al máximo.

¿Qué hacemos para ajustar la implementación?

Para ello vamos a eliminar un for y hacer las operaciones directamente con las filas.

```
void Image::ShuffleRows() {
   const int p = 9973;
   Image temp(rows,cols);
   int newr;
   for (int r=0; r<rows; r++){
```

```
        newr = r*p % rows;
        temp.img[r]=img[newr];
    }
    Copy(temp);
}
```

5.Comparar la eficiencia experimental de la nueva implementación ajustada con la anterior.

Ahora ejecutamos este comando:

./barajar output/zoom_shuffle.pgm output/baraja_99pixeles.pgm

y nos da como salida:

Fichero origen: output/zoom_shuffle.pgm
Fichero resultado: output/baraja_99pixeles.pgm

Dimensiones de output/zoom_shuffle.pgm:
  Imagen = 99 filas x 99 columnas
La imagen se guardo en output/baraja_99pixeles.pgm
Ticks de reloj : 291
Segundos : 0.000291

Donde podemos ver que tarda menos tiempo que con la implementación anterior, sin hacer el ajuste de ShuffleRows() nos salía más tiempo pero una vez hecho nos da un tiempo menor.

6.Generación aleatoria de imágenes, así como la representación de sus gráficas en GnuPlot.

Para ellos vamos a modificar el anterior programa para que nos muestre en una columna el size de la imagen y en otra columna el tiempo que tarda en ejecutar el ShuffleRows. Esto lo hacemos para crear un archivo de datos.txt que nos facilite la representación con gnuplot.

Para ello hemos ido más allá. Hemos automatizado todos los procesos mediante scripts, los cuales vamos explicando en ManualUso.txt que se encuentra en la carpeta <Ejercicio 5 Práctica Imágenes>.