Práctica Final: Descubre Objetivos

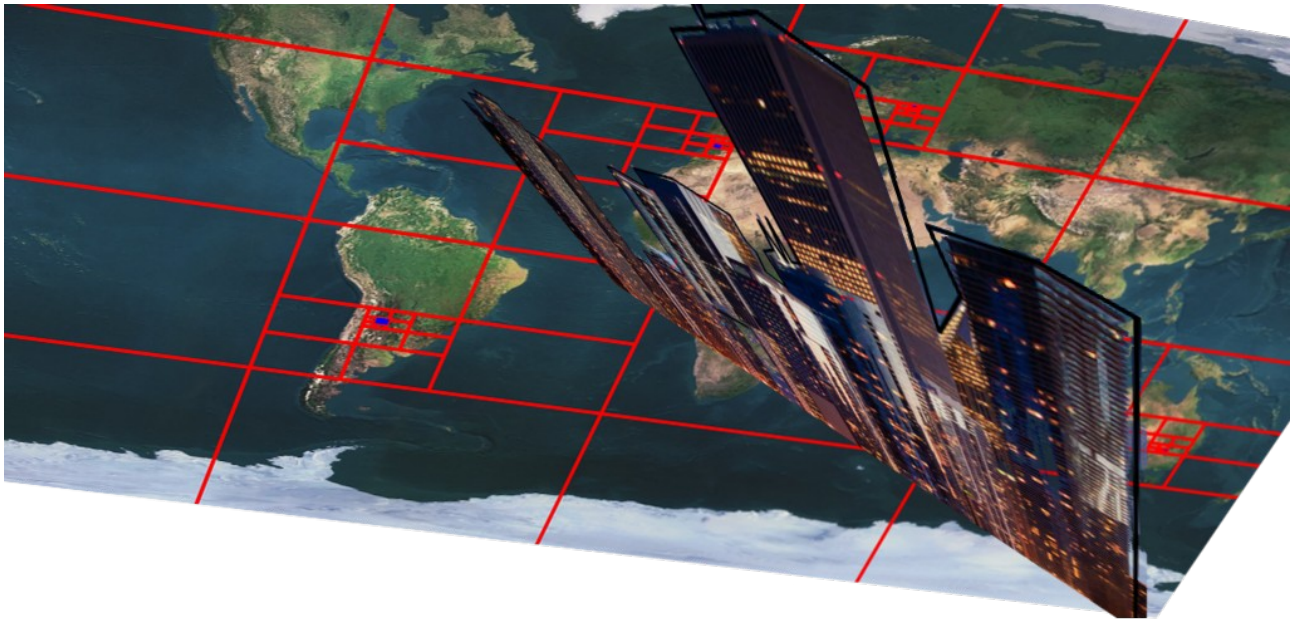
Dpto. Ciencias de la Computación e Inteligencia Artificial
E.T.S. de Ingenierías Informática y de Telecomunicación
Universidad de Granada

DECSAI

Estructuras de Datos

Grado en Ingeniería Informática C

Índice de contenido



1. Introducción

Los objetivos de este guión de prácticas son los siguientes:

- Resolver un problema eligiendo la mejor estructura de datos para las operaciones que se solicitan

Los requisitos para poder realizar esta práctica son:

1. Haber estudiado el Tema 1: Introducción a la eficiencia de los algoritmos
2. Haber estudiado el Tema 2: Abstracción de datos. Templates.
3. Haber estudiado el Tema 3: T.D.A. Lineales.
4. Haber estudiado el Tema 4: STL e Iteradores.
5. Haber estudiado estructuras de datos jerárquicas: Árboles

2. Objetivo.

El objetivo de esta práctica es llevar a cabo el análisis, diseño e implementación de un proyecto. Con tal fin el alumno abordará un problema donde se requiere estructuras de datos que permiten almacenar grandes volúmenes de datos y poder acceder a ellos de la forma más eficiente.

3. Ejercicio

Desde el centro de inteligencia español se sabe que varios países son objetivos de bandas criminales. Nuestro servicio de espías han proporcionado las ciudades de posibles atentados por estas bandas. Así que nuestro objetivo será obtener estas ciudades para aumentar la vigilancia.

El problema es que los espías proporcionan la información codificada en un quadtree.

Un quadtree es un árbol cuaternario que tiene 0 o 4 hijos.

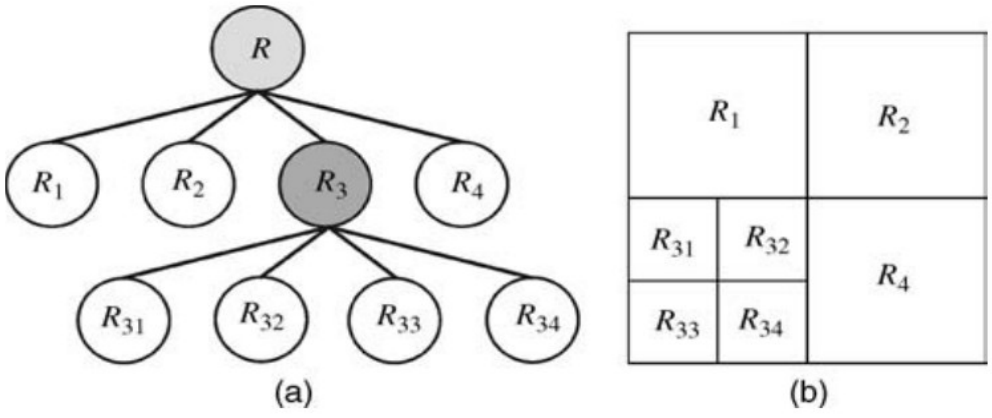


Figura 1: Ejemplo de quadtree (a). En (b) partición de una imagen asociada a la estructura quadtree en (a).

Podemos asociar un Quadtree a una imagen. Así el nodo raíz del Quadtree hace referencia a la

imagen total. Si el nodo raiz tienen cuatro hijos significa que la imagen se divide en cuatro subimagenes iguales. Así recursivamente para cualquier otro nodo. En la Figura 1 se puede observar un quatree a la izquierda y la partición realizada sobre una imagen. Supongamos que la Figura1(b) es una imagen del mapa mundo (como se muestra en la Figura 2).Si la imagen dada en la Figura1(b) tuviese dimensiones 256 filas x 128 columnas, las regiones generadas se pueden describir con cuatro parámetros: fila y columna de la esquina superior izquierda y fila y columna de la esquina inferior derecha. Así para la Figura1(b) tenemos:

- $\mathbf{R_1}$ (0,0,128,64) , $\mathbf{R_2}$ (0,64,128,128)
- $\mathbf{R_{31}}$ (128,0,192,32), $\mathbf{R_{32}}$ (128,32,192,64), $\mathbf{R_{33}}$ (192,0,256,32), $\mathbf{R_{34}}$ (192,32,256,64)
- $\mathbf{R_4}$ (128,64,256,128)

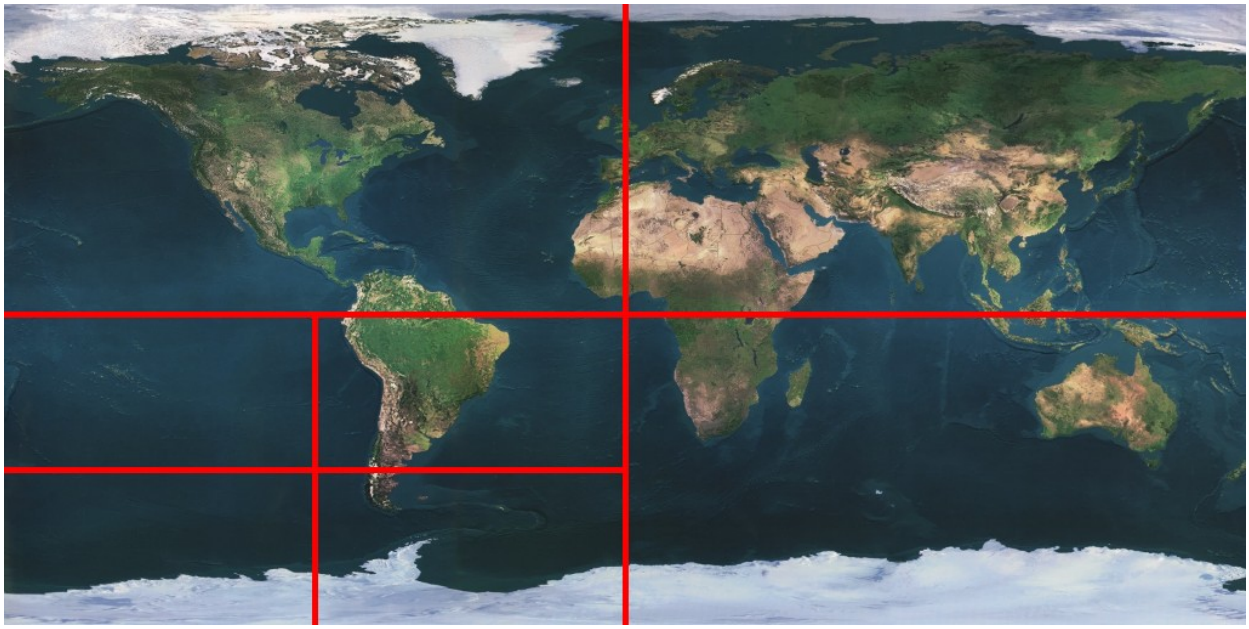


Figura 2: Ejemplo de un quadtree sobre una imagen del mapa mundi

Para obtener la partición dada en la imagen de la Figura 1(b) el espía podría indicarnos con ***1*** si el trozo de imagen que estamos analizando tenemos que dividirlo en cuatro partes iguales o no (con un ***0*** se indica que no tenemos que dividirlo). Si no tenemos que dividirlo el espía nos tiene que indicar si dicha región es un objetivo, con un ***1***, o no lo es con un ***0.***

Por ejemplo suponed que en la Figura 1(b) la región donde se puede producir un posible atentado es la R_{34} . Por lo tanto el código que nos enviaría el espía sería:

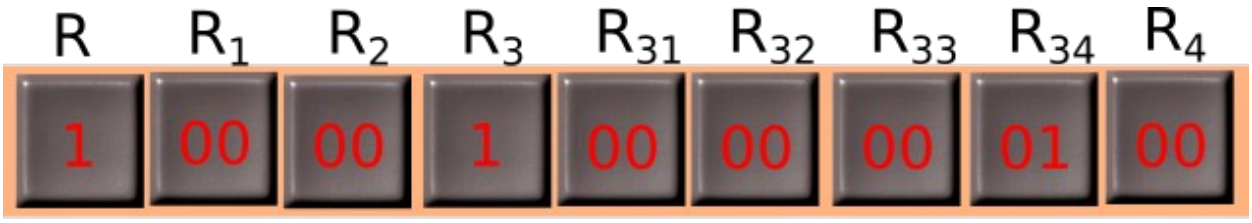


Figura 3: Secuencia para codificar la imagen de la Figura1(b) y 2 con objetivo en la region R_{34}

Por otro lado disponemos de un fichero con información de las ciudades del mundo. Como hemos visto en la Práctica 2 y 3, cada ciudad se describre como:

1. Nombre
2. País
3. Latitud geográfica. En el rango [-90º,90º]
4. Longitud geográfica. En el rango [-180º,180º]
5. Población

Para descubrir el nombre de las ciuades de posibles objetivos tenemos que asociar a un conjunto de píxeles en la imagen del mapamundi coordenadas geográficas dadas por latitud y longitud. De esta forma se establece que ciudad (o ciudades) están en ese rango de posiciones geográficas.

Así dada una posición en el mapamundi (indicada por un par (fila,columna)), podemos obtener su posición geográfica dada por latitud y longitud como:

$$latitud = 90 - \frac{fila * 180}{totalfilas} \quad \text{y} \quad longitud = \frac{360 * columna}{totalcolumnas} - 180$$

Siguiendo nuestro ejemplo en el que nuestro objetivo está en la región R_{34} (ver Figura 1.(b)) , como esta tiene como esquina superior izquierda por (fila,columna) (192,32) y esquina inferior derecha (256,64), aplicando las dos fórmulas anteriores obtenemos como valores de latitud y longitud para (192,32)

$$latitud = 90 - \frac{192 * 180}{256} = -45 \quad \text{y} \quad longitud = \frac{360 * 32}{128} - 180 = -90$$

De la misma forma para la esquina inferior derecha (256,64) se obtiene un valor de latitud=-90 y longitud =0.

Con estos valores buscamos todas las ciudades con latitud en el rango (-90,-45) y longitud en el rango (-90,0). Siendo las ciudades en este rango los posibles objetivos de atentandos.

4. Programa Descubre_Objetivos

Este programa obtiene una lista de ciudades objetivo, codificadas en un quadtree. Un ejemplo de llamada a este programa desde la línea de órdenes sería:

```
prompt% bin/descubre_objetivos datos/ciudades.csv datos/codigo_objetivos_ciudades1.txt
datos/mapa1.ppm salida.ppm
```

Los parámetros de entrada son los siguientes:

1. El nombre del fichero con la información de todas las ciudades del mundo
2. El nombre del fichero con la secuencia que codifica el quadtree y almacena las ciudades objetivo (como se ha descrito en la seccion 3).
3. El nombre del fichero con la imagen del mapa mundi
4. El nombre del fichero con la imagen representando la particion Quadtree.

La ejecución de este comando daría lugar a la siguiente salida:

```
Granada;Spain
Moscow;Russia
Monte Quemado;Argentina
Maitland;Australia
Newcastle;Australia
```

Y la imagen resultante es la que se muestra en la Figura 4

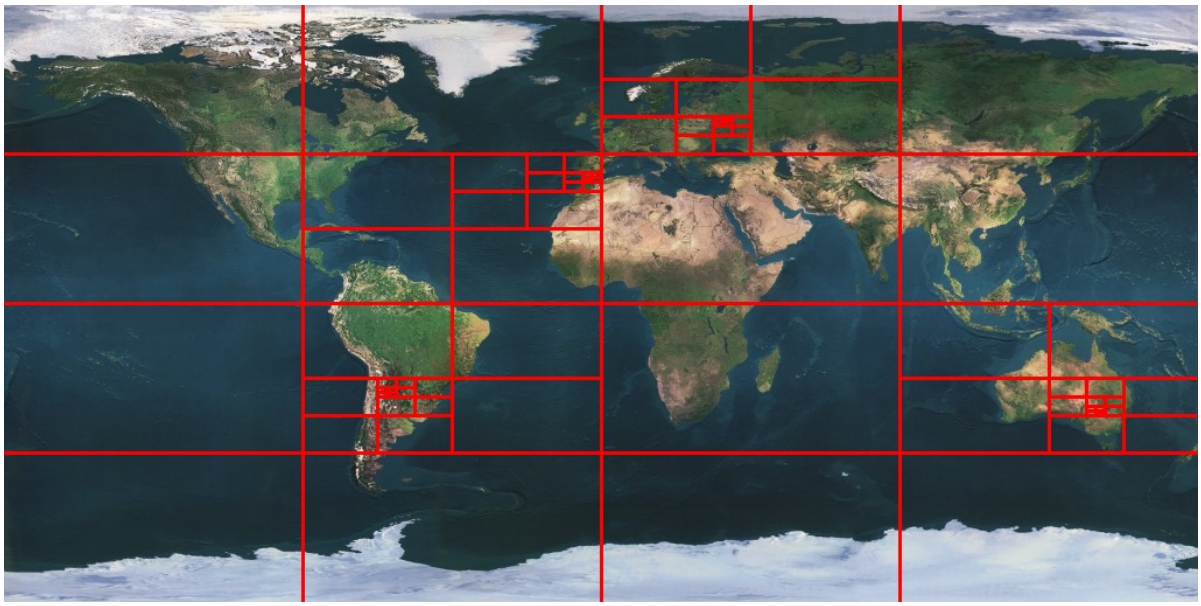


Figura 4: Particion Quadtree sobre el mapamudi para codigos_objetivos_ciudades1.txt.

5. Tareas a realizar

Antes de aborar el programa descrito en la sección 4 el alumno/a deberá desarrollar diferentes tipos de datos y realizar los test correspondientes. Así las tareas a realizar son:

1. Desarrollar el TDA Quadtree.
2. Probar el TDA Quadtree con el fichero test_quadtree.cpp
3. Desarrollar el TDA Imagen
4. Probar el TDA Imagen con el fichero test_imagen.cpp
5. Recuperar el TDA Ciudad, Poiscion y Ciudades desarrollado en la práctica 3.
6. Construir el programa descubre_objetivos descrito en la sección 4

5.1. TDA Quadtree.

Un objeto de tipo Quadtree es un árbol cuartenario, es decir, cada nodo tiene 0 o cuatro hijos.

Para desarrollar este nuevo TDA vamos a usar el TDA ArbolGeneral dado en el material. La interfaz de TDA Quadtree también la podéis encontrar en el material dado al alumno. Así la representación de un quadtree es la siguiente:

```
//FICHERO Quadtree.h
class Quadtree{
….
  private:
     ArbolGeneral<pair<bool,char> > tree;
….
  public:
  ...
       class iterator {
       private:
         ArbolGeneral<pair<bool,char> >::iter_preorden it;
       public:

       };
       class const_iterator {
       private:
         ArbolGeneral<pair<bool,char> >::const_iter_preorden it;
       public:
         const_iterator(){}
        ...
       };

...
       iterator begin();

       iterator end();

       const_iterator begin()const;

       const_iterator end()const ;

       friend istream& operator>>(istream& in, Quadtree & Q);

       friend ostream& operator<< (ostream& out, const Quadtree & Q);

};
```

```
//FIN DE Quadtree.h
```

Como se puede observar un objeto de Quadtree tiene asociado dos iteradores un constante y otro no constante. Estos iteradores permiten iterar por el árbol en preorden.

5.2. Probando el TDA Quadtree: test_quadtree.

El fichero test_quadtree.cpp permitirá probar nuestro TDA Quadtree. Para ello tiene cuatro secciones distinguidas:

1. Sección 1.- Prueba la construcción de diferentes quadtrees. También prueba el funcionamiento de los operadores de entrada y salida de un Quadtree.
2. Sección 2.- Comprueba que los métodos de inserción y borrado de un quatree funciona correctamente.
3. Sección 3.- Comprueba que las operaciones de comparación de dos quadtree están bien implementadas.
4. Sección 4.- Comprueba que los iteradores no constante y constante funcionan.

Un ejemplo de llamada a el ejecutable test_quadtree puede ser el siguiente:

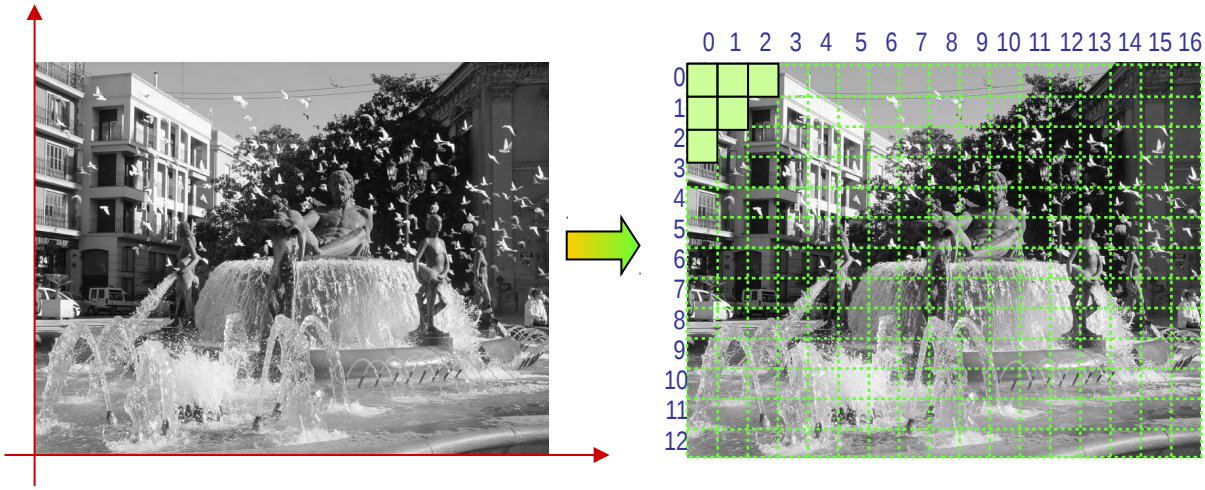
```
prompt% bin/test_quadtree datos/codigo_objetivos_ciudades1.txt
```

5.3. T.D.A Imagen

Una imagen digital se puede ver como una matriz donde cada casilla de la matriz almacena un píxel. El contenido de cada casilla de la matriz (o píxel) dependerá del uso que se le de a la imagen. Algunos ejemplos podría ser:

- Una imagen para señalar los puntos donde se encuentra la información de un objeto en otra imagen. Por ejemplo una imagen máscara de un avión que mediante dos valores indica donde se encuentra el avión si la máscara toma valor 255 o 0 en caso contrario.

- Si queremos almacenar una escena en blanco y negro, podemos crear un rango de valores de luminosidad (valores de gris), por ejemplo los enteros en el rango [0,255] (el cero es negro, y el 255 blanco). En este caso cada casilla de la matriz almacena un byte.
- Si queremos almacenar una escena con información de color, podemos fijar en cada celda una tripleta de valores indicando el nivel de intensidad con el que contribuyen 3 colores básicos (*rgb red-green-blue*) para formar el color requerido.

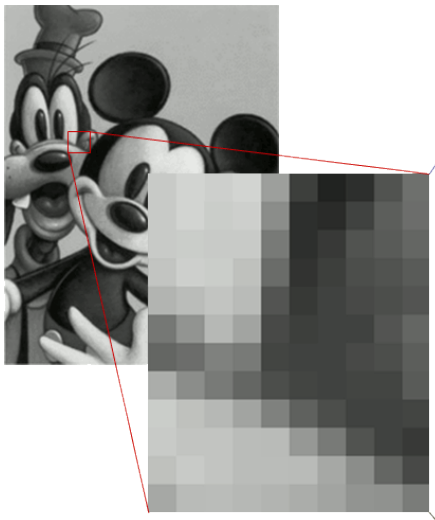


5.3.1. Imágenes en blanco y negro.

Para representar las imágenes en blanco y negro podemos usar un rango de valores para indicar todas las tonalidades de gris que van desde el negro hasta el blanco. Las imágenes almacenarán en cada píxel un valor de gris desde el 0 al 255. Por ejemplo, un píxel con valor 128 tendrá un gris intermedio entre blanco y negro. La elección del rango [0,255] se debe a que esos valores son los que se pueden representar en un byte(8 bits). Por tanto, si queremos almacenar una imagen de niveles de gris, necesitaremos ancho·alto bytes. En el ejemplo de la imagen anterior, necesitaríamos 64Kbytes para representar todos sus píxeles.

A modo de ejemplo, en la siguiente figura mostramos una imagen digital de niveles de gris que tiene 320 filas y 244 columnas. Cada uno de los 320×244 = 78080 puntos contiene un

valor entre 0 (visualizado en negro) y 255 (visualizado en blanco). Si analizamos un trozo de la imagen de 10×10 filas podemos ver los valores de luminosidad con más detalle.



202	211	208	211	158	62	34	47	82	112
202	211	204	208	140	47	42	66	93	109
202	204	202	202	121	47	57	82	90	101
195	208	204	192	107	51	62	77	82	90
179	188	195	186	84	55	66	73	77	84
119	148	183	163	73	62	66	66	84	101
101	109	128	121	68	66	66	73	77	93
173	140	121	101	77	68	66	68	68	90
204	192	183	163	128	96	77	66	66	68
202	195	195	188	186	152	121	82	66	57
192	202	195	188	188	186	167	140	101	73
167	186	188	183	179	173	163	148	146	123

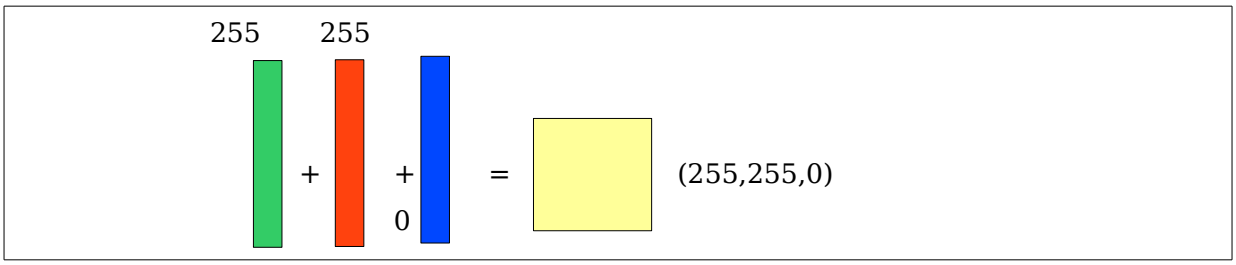
5.3.2. Imágenes en color

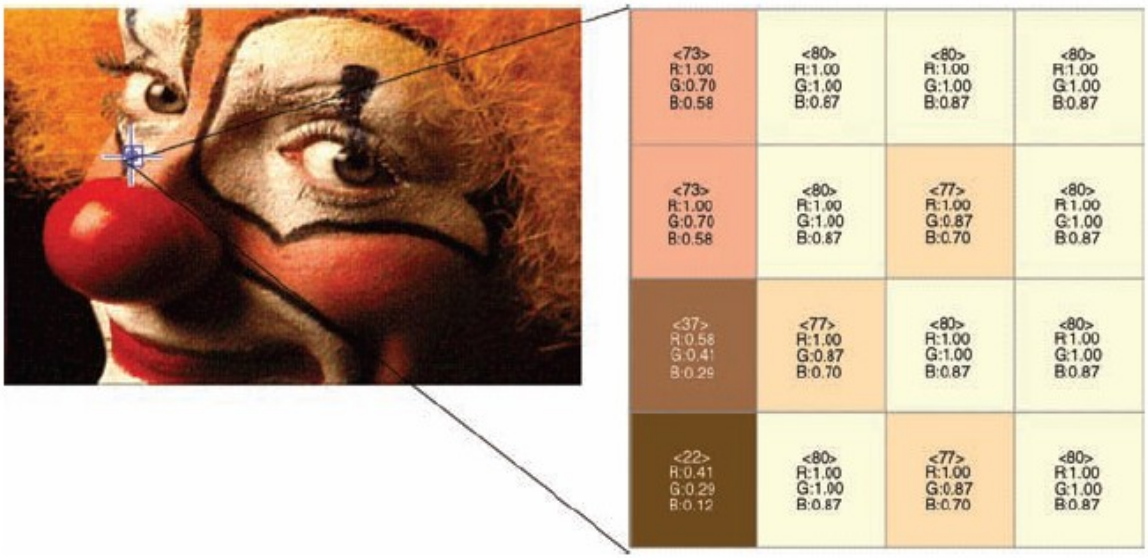
Para representar un color de forma numérica, no es fácil usar un único valor, sino que se deben incluir tres números. Existen múltiples propuestas sobre el rango de valores y el significado de cada una de esas componentes, generalmente adaptadas a diferentes objetivos y necesidades.

En una imagen en color, el contenido de cada píxel será una tripleta de valores según un determinado modelo de color. En esta práctica consideraremos el modelo RGB. Este modelo es muy conocido, ya que se usa en dispositivos como los monitores, donde cada color se representa como la suma de tres componentes: rojo, verde y azul.

Podemos considerar distintas alternativas para el rango de posibles valores de cada componente, aunque en la práctica, es habitual asignarle el rango de números enteros desde el 0 al 255, ya que permite representar cada componente con un único byte, y la variedad de posibles colores es suficientemente amplia. Por ejemplo, el ojo humano no es capaz de distinguir un cambio de una unidad en cualquiera de las componentes.

En la siguiente figura se muestra un ejemplo en el que se crea un color con los valores máximos de rojo y verde, con aportación nula del azul. El resultado es el color (255,255,0), que corresponde al amarillo.





Ejemplo de una imagen a color y un zom sobre un pequeña región. Aunque los valores R,G,B que se muestran estan en el rango [0,1]. Realmente multiplicando por 255 obtenemos los valores como nosotros los vamos a tratar en el rango [0,255].

5.3.3. Funciones de E/S de imágenes

Las imágenes que manejaremos están almacenadas en un fichero que se divide en dos partes:

1. **Cabecera**. En esta parte se incluye información acerca de la imagen, sin incluir el valor de ningún píxel concreto. Así, podemos encontrar valores que indican el tipo de imagen que es, comentarios sobre la imagen, el rango de posibles valores de cadapíxel, etc. En esta práctica, esta parte nos va a permitir consultar el tipo de imagen y sus dimensiones sin necesidad de leerla.
1. **Información**. Contiene los valores que corresponden a cada píxel. Hay muchas formas para guardarlos, dependiendo del tipo de imagen de que se trate, pero en nuestro caso será muy simple, ya que se guardan todos los bytes por filas, desde la esquina superiorizquierda a la esquina inferior derecha.

Los tipos de imagen que vamos a manejar serán PGM (Portable Grey Map file format) y PPM (Portable Pix Map file format), que tienen un esquema de almacenamiento con cabecera seguida de la información, como hemos indicado. El primero se usará para las imágenes en blanco y negro y el segundo para las imágenes en color.

Para simplificar la E/S de imágenes de disco, se facilita un módulo (archivo de cabecera y de definiciones), que contiene el código que se encarga de resolver la lectura y escritura de ambos formatos. Por tanto, el alumno no necesitará estudiar los detalles de cómo es el formato interno de estos archivos. En lugar de eso, deberá usar las funciones proporcionadas para resolver ese problema. El archivo de cabecera contiene lo siguiente:

```
#ifndef _IMAGEN_ES_H_
#define _IMAGEN_ES_H_

enum TipoImagen {IMG_DESCONOCIDO, ///< Tipo de imagen desconocido
                 IMG_PGM, ///< Imagen tipo PGM
                 IMG_PPM ///< Imagen tipo PPM
};
TipoImagen LeerTipoImagen (const char nombre[], int& filas, int& columnas);
bool LeerImagenPPM (const char nombre[], int& filas, int& columnas,
unsigned char buffer[]);
bool EscribirImagenPPM (const char nombre[], const unsigned char datos[],int f, int c);
bool LeerImagenPGM (const char nombre[], int& filas, int& columnas, unsigned char buffer[]);
bool EscribirImagenPGM (const char nombre[], const unsigned char datos[],int f, int c);
#endif
```

5.4. Fichero imagen.h

En el fichero imagen.h, dado en el material se ha dado la representación de una imagen y las funcionalidades o métodos que deben tener. La implementación de lo métodos de imagen se harán en el fichero imagen.cpp. Al abrir este fichero se verá que se ha dado implementadas los métodos EscribirImagen y LeerImagen.

6. Probando imagen: test_imagen

Para probar nuestro TDA imagen haremos uso de test_imagen.cpp dado en el material. Este fichero usa el fichero pintaformas (pintaformas.h y pintaformas.cpp) en el que se han definido funciones para pintar sobre una imagen lineas, cuadrados o ciculos.

Por lo tanto para testear nuestra imagen lo primero que haremos es leer una imagen de disco. A continuacion Pintaremos un cuadrado y dos lineas. Finalmente escribiremos la imagen en disco.

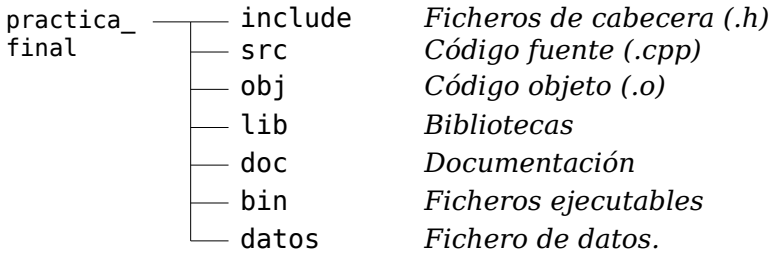
Un ejemplo de llamada para test_imagen puede ser:

```
prompt% bin/test_imagen datos/mapa1.ppm salida.ppm
```

7. Práctica a entregar

El alumno deberá empaquetar todos los archivos relacionados en el proyecto en un archivo con nombre “practica_final.*tgz*” y entregarlo antes de la fecha que se publicará en la página web de la asignatura. Tenga en cuenta que no se incluirán ficheros objeto, ni ejecutables, ni la carpeta datos. Es recomendable que haga una “limpieza” para eliminar los archivos temporales o que se puedan generar a partir de los fuentes.

El alumno debe incluir el archivo *Makefile* para realizar la compilación. Tenga en cuenta que los archivos deben estar distribuidos en directorios*:*



Para realizar la entrega, en primer lugar, realice la limpieza de archivos que no se incluirán en ella, y sitúese en la carpeta superior (en el mismo nivel de la carpeta "*practica_final*") para ejecutar:

```
prompt% tar zcv practica_final.tgz practica_final
```

tras lo cual, dispondrá de un nuevo archivo practica_final.tgz que contiene la carpeta letras así como todas las carpetas y archivos que cuelgan de ella.