Práctica final: Rutas aéreas

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

Estructuras de Datos

Doble Grado en Ingeniería Informática y Matemáticas

Índice

1. Introducción

El objetivo de esta práctica final de la asignatura consiste en resolver un problema concreto eligiendo adecuadamente las estructuras de datos que se utilizarán para resolver dicho problema.

Individualmente o por parejas, se desarrollará una solución que, utilizando las estructuras de datos adecuadas, permita gestionar grandes volúmenes de datos y garantizar el acceso a los datos de la forma más eficiente posible.

2. Descripción

En esta práctica, realizaremos el análisis, diseño e implementación de un pequeño proyecto.

Se desea diseñar un conjunto de programas para una compañía aérea. Uno de estos programas se utilizará para visualizar las rutas de la compañía aérea sobre un mapa. Por ejemplo, puede que queramos visualizar en un mapa del mundo la ruta con código R3. El programa creará una imagen como la de la figura 1 y, al mismo tiempo, mostrar por pantalla la lista de países en los que se hace escala (en este caso, Canadá, Estados Unidos, Perú, España, China, Australia, Japón y Rusia).

En la visualización de la ruta aérea, tal como se muestra en la figura, se incluirá la bandera de los países en los que se hace escala y un icono del avión orientado según la línea recta que une los dos aeropuertos que forman un segmento de la ruta realizada.

NOTA: Se puede hacer uso de la STL.

Figura 1: Visualización de una ruta sobre el mapa. En este caso, la ruta es (Canadá, Estados Unidos, Perú, España, China, Australia, Japón, Rusia)

2.1. Tareas

Cada alumno (o pareja de alumnos(debe llevar a cabo las siguientes tareas:

1. Dar una especificación de los T.D.A. que considere necesarios, incluyendo los usados en la práctica 3:
   - Punto
   - Ruta
   - Almacén de Rutas.
2. Definir el conjunto de operaciones de cada T.D.A. con su especificación.
3. Comprobar su implementación de las distintas operaciones con programas de prueba.
4. Construir los programas que se describen en los siguientes apartados.

2.1.1. Rotar una imagen

Este programa permitirá al usuario, dada una imagen y un ángulo en grados, obtener una nueva imagen que es la versión rotada de la imagen original. Este programa deberá funcionar desde la línea de comandos de la siguiente forma:

```
rota avion.ppm 45 avion_45.ppm
```

En esta llamada, “avion.ppm” indica el fichero que contiene la imagen original y 45 es el ángulo de rotación 45, mientras que “avion_45.ppm” es el nombre de la imagen de salida.

Ambas imágenes estarán en formato PPM, un formato descrito en las secciones 2.2.3 y 2.3.4 de este guión.

Figura 2: Una imagen rotada con diferentes ángulos.

Para realizar la rotación, el alumno tiene a su disposición el código que se encuentra en el fichero pruebarotacion.cpp, así como el módulo imagenES para realizar la lectura y escritura de imágenes en formato PPM.

En el fichero pruebarotacion.cpp está implementada la función que realiza la rotación (Rota). En esta función, se aplica la siguiente matriz de rotación a cada pìxel (i,j) de la imagen:

$$\begin{pmatrix} \cos(\alpha) & \sin(\alpha) \\ -\sin(\alpha) & \cos(\alpha) \end{pmatrix}$$

2.1.2. Superponer imágenes

Este programa permitirá al usuario, dada una imagen de fondo y una segunda imagen, superponer la segunda imagen sobre la imagen de fondo. El comando que se deberá ejecutar desde la línea de comandos es el siguiente:

```
pega espana_reshigh.ppm avion2.ppm mascara_avion2.pgm españa_avion2_blending.ppm 0 0 1
```

Los parámetros de este programa son los siguientes:

1. Nombre de la imagen de fondo en formato ppm.
2. Nombre de la imagen que se va a pegar sobre la primera (en formato ppm)
3. Nombre de la imagen que se usará como máscara de la segunda imagen.
4. Nombre de la imagen de salida.
5. Fila y columna a partir de la cual se realiza el pegado.
6. Un valor *0* si la segunda imagen oculta completamente la imagen de fondo (pegado opaco) ó *1* si la imagen superpuesta se combina al combinarse con la imagen de fondo (pegado transparente o “blending”) .

Los resultados obtenidos al superponer imágenes utilizando distintos valores posibles valores de opacidad (0 ó 1) se muestran en la figura 3

El algoritmo de pegado se describe a continuación:

Sean If la imagen de fondo, I2 la imagen superpuesta, M la imagen de máscara, Ir la imagen resultante y (posi,posj) la posición de la imagen de fondo donde se comienza a pegar.

Para cada pixel (i,j) en I2, obtener Ir como

$$Ir(posi+i, posj+j) = \begin{cases} I2(i,j) & \text{si } pegado=OPACO \text{ y } M(i,j)=255 \\ (I2(i,j)+If(i,j))/2 & \text{si } pegado=BLENDING \text{ y } M(i,j)=255 \end{cases}$$

NOTA: Una imagen de máscara es una imagen en blanco y negro que sólo tiene dos niveles de gris (0 y 255). Un valor 0 indica que ese píxel no será tenido en cuenta en el proceso de superposición, mientras que el valor 255 indica que ese píxel sí forma parte de la imagen superpuesta que se mostrará sobre la imagen de fondo. Estas máscaras se almacenan en formato PGM (véanse las secciones 2.2.3 y 2.3.4).

2.1.3. Mostrar rutas aéreas

Este tercer programa se utilizará para obtener un mapa del mundo sobre el mostrará la ruta elegida. La figura 1 muestra un ejemplo de lo que se pretende obtener.

El usuario indicará un fichero que incluye datos de países (ver la especificación del fichero de países en la sección 2.3.2), un almacén de rutas (tal como se construyó en la práctica 3), una imagen del mapa del mundo, una imagen del avión como la mostrada en la figura 2 y su máscara. Además, hará falta indicar el directorio donde se almacenan las banderas de cada país (dir_banderas en el siguiente ejemplo) y la ruta que queremos visualizar.

Desde la línea de comandos, el programa se invocará de la siguiente manera:

```
ruta  paises.txt mapa.ppm dir_banderas almacen_rutas.txt avion.ppm mascara_avion.pgm R3
```

En primer lugar, el programa mostrará por pantalla los detalles de la ruta indicada (p.ej. R3). Tras esto, el programa creará en disco una imagen PPM cuyo nombre será el código de la ruta escogida con extensión .ppm. En la imagen aparecerá la bandera de los países en los que se hace escala y, para cada segmento de la ruta, se dibujará el avión al comienzo, en su punto intermedio y al final (con la orientación adecuada para proseguir la ruta.

La traslación de un par (latitud,longitud) a un píxel (i,j) de la imagen se hace de la siguiente forma:

$$columna = (totalcolumnas/360.0)*(180+longitud)$$
$$fila = (totalfilas/180.0)*(90-latitud)$$

donde *totalcolumnas* y *totalfilas* son el número de filas y número de columnas de la imagen con el mapa del mundo.

Imagen de fondo $I_f(i,j)$

Imagen superpuesta $I_2(i,j)$

Máscara de imagen M(i,j)

Pegado opaco $I_r(i,j)$

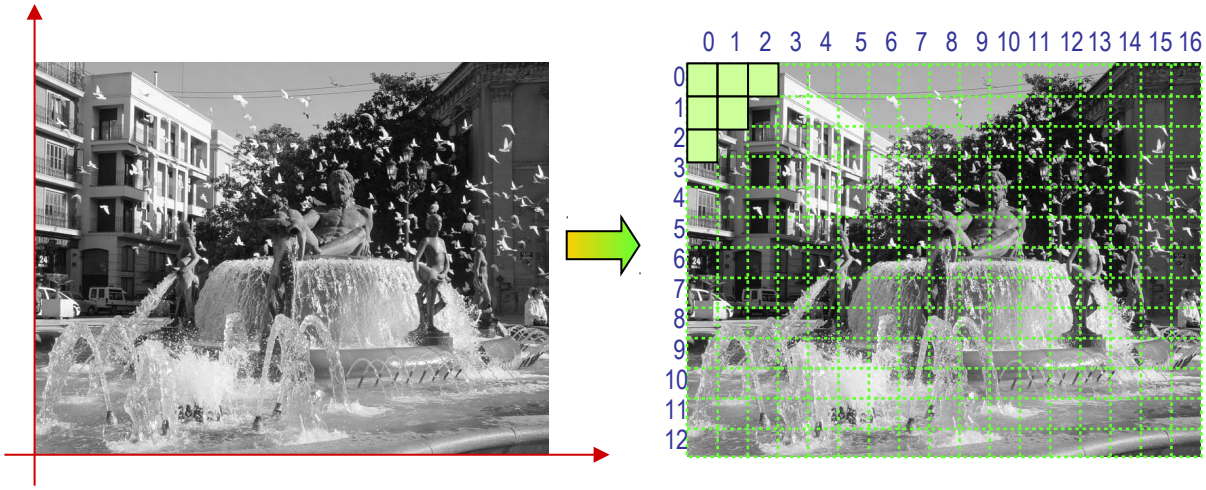
Pegado transparente $I_r(i,j)$

Figura 3: Resultados del pegado de una imagen sobre otra.

2.2. Imágenes

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:

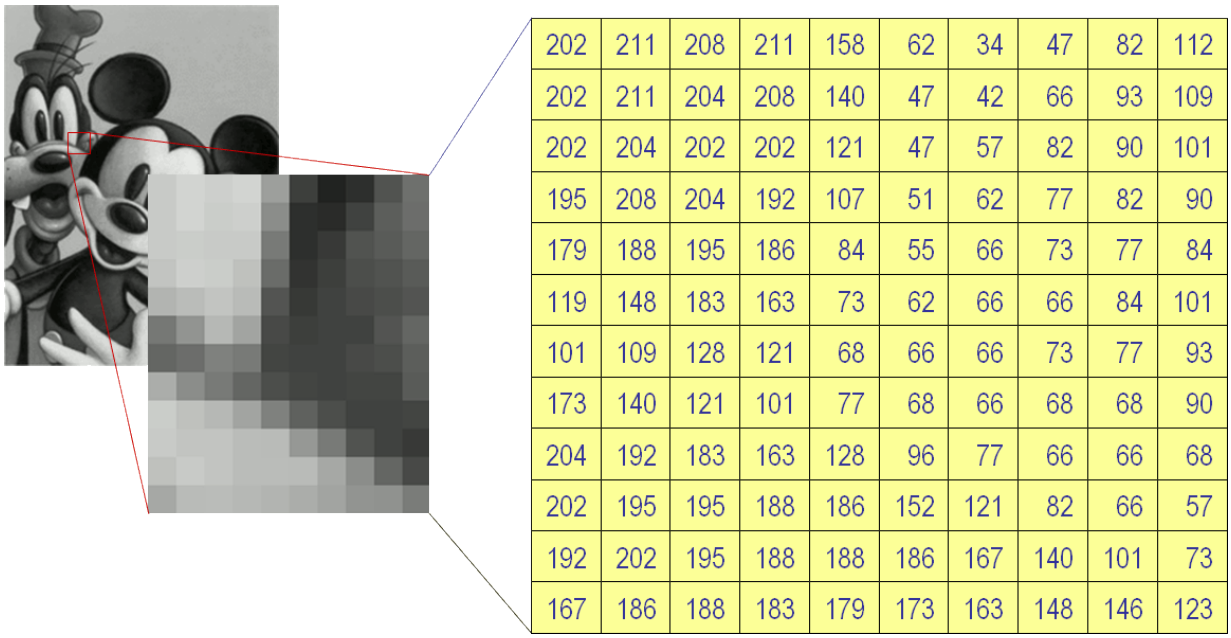
- Una imagen para señalar los puntos donde se encuentra la información de un objeto en otra imagen. Por ejemplo una imagen máscara de un avión que mediante dos valores indica donde se encuentra el avión si la máscara toma valor 255 o 0 en caso contrario.
- Si queremos almacenar una escena en blanco y 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.
- Si mezclamos el primer y tercer ejemplo podríamos querer almacenar una escena junto con la información de donde se encuentra el objeto en la escena. Así almacenaríamos información para los tres colores básicos (cada color en el rango [0,255]) e información para saber si un píxel forma parte del objeto con dos valores 0,si no forma parte, o 255 en caso contrario.



2.2.1. 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

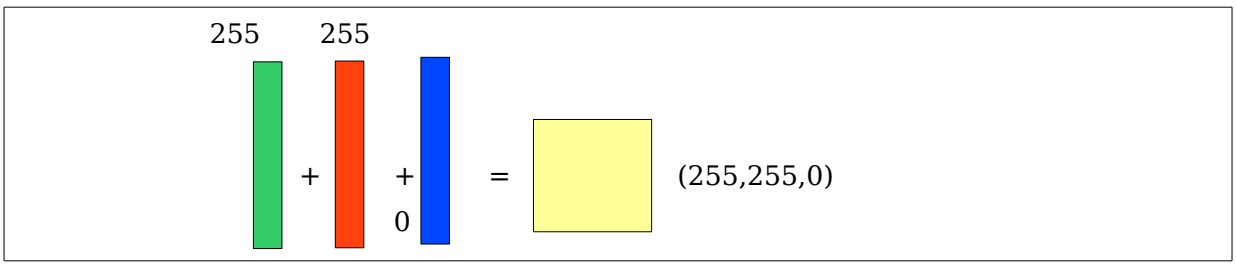
2.2.2. Imágenes en color

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.



2.2.3. 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
```

Además, se incluye documentación en formato doxygen para que sirva de muestra y pueda ser usada como referencia para estas funciones. Ejecute “make documentacion” en el paquete que se le ha entregado para obtener la salida de esa documentación en formato HTML (use un navegador para consultarla). Si estudia detenidamente las cabeceras de las funciones que se proporcionan, verá que con el nombre del parámetro, así como con su carácter de entrada o salida, es fácil intuir el objetivo de cada uno de ellas. Tal vez, la parte más confusa pueda surgir en los parámetros correspondientes al buffer o los datos de la imagen (vectores de unsigned char):

1. Si la imagen es PGM -de grises- será un vector que contenga todos los bytes consecutivos de la imagen. Así, la posición 0 del vector tendrá el píxel de la esquina superior izquierda, la posición 1 el de su derecha, etc.
2. Si la imagen es PPM -de color- será un vector similar. En este caso, la posición 0 tendrá la componente R de la esquina superior izquierda, la posición 1 tendrá la poción G, la posición 2 la B, la posición 3 la componente R del siguiente píxel, etc. Es decir, añadiendo las tripletas RGB de cada píxel.

2.3. Ficheros

2.3.1. Almacén de rutas

Un ejemplo de fichero de almacén de rutas es el siguiente:

```
#Rutas
R1 5 (34.520418555522845,69.20082090000005) (52.50786264022465,13.426141949999987)
(7.406652727545182,12.344585699999925) (-0.18659558628491132,-78.4305382)
(40.437944725164726,-3.6795366500000455)
R2 8 (58.695433501291085,-96) (35.08690549340541,-103.72339606166992)
(-12.055345316962327,-77.04518530000001) (40.437944725164726,-3.6795366500000455)
(37.943768420529985,104.13611175000005) (-27.787075486256633,133.28132295)
(35.673473752079516,139.71038800000008) (62.88647107195116,61.551173617626986)
R3 5 (17.246400332673307,-19.670602940234403) (4.283635422564345,-74.22403995000002)
(51.528868434293244,  -0.10159864999991441) (62.88647107195116,61.551173617626986)
(37.943768420529985,104.13611175000005)
R4 11 (14.422538164676899,-87.63432239999997) (48.85887766623369,2.3470598999999766)
(24.725939314861463 ,46.822528799999986) (58.695433501291085,-96)
(35.08690549340541,-103.72339606166992) (-12.055345316962327,-77.04518530000001)
(40.437944725164726,-3.6795366500000455) (37.943768420529985,104.13611175000005)
(-27.787075486256633,133.28132295) (35.673473752079516,139.71038800000008)
(62.88647107195116,61.551173617626986)
R5 5 (20.669378555990964,-105.20813450000003) (-19.051901092806112,29.15280180000002)
(-34.61590069251671,-58.433298449999995) (58.695433501291085,-96)
(20.669378555990964,-105.20813450000003)
```

El formato del fichero es el siguiente:

1. La palabra mágica #Rutas
2. A continuación, aparece descrita cada una de las rutas en un línea:
    - En primer lugar, aparece el código de la ruta.
    - A continuación, el número de puntos de la ruta.
    - Por último, tantos puntos geográficos separados por “blancos” (espacios, tabuladores...). Cada punto se especifica con dos valores reales que indican la latitud y longitud del punto. Dichos valores se especifican entre paréntesis y separados por una coma.
3. Después de las rutas, pueden aparecer (o no) puntos de interés. Si se especifican, aparecerá la palabra mágica #Puntos_de_Interes en un línea. A continuación, en las siguientes líneas, se especificarán los puntos de interés junto con su descripción, tal como se indicaba en la práctica 3.

2.3.2. Países

Un ejemplo de fichero de países:

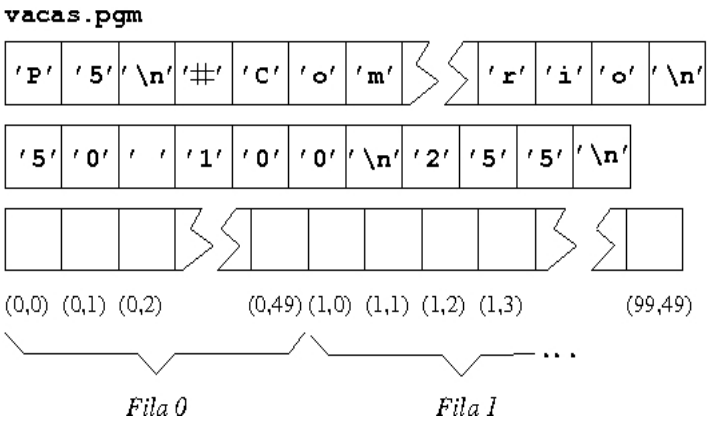
```
# Latitud             Longitud              Pais           Bandera
34.520418555522845    69.20082090000005     Afganistan     afganistan.ppm
41.332136072796175    19.812877200000003    Albania        albania.ppm
52.50786264022465     13.426141949999987    Alemania       alemania.ppm
-11.294616098942507   17.877003150000064    Angola         angola.ppm
24.725939314861463    46.822528799999986    ArabiaSaudita  arabiasaudi.ppm
28.415101049232497    1.6666662999999744    Argelia        argelia.ppm
-34.61590069251671    -58.433298449999995   Argentina      argentina.ppm
40.0817343141965      45.04071690000001     Armenia        armenia.ppm
-27.787075486256633   133.28132295          Australia      australia.ppm
47.71329162782909     13.34573480000006     Austria        austria.ppm
40.17498736058696     47.56637439999997     Azerbaiyan     azerbaiyan.ppm
```

El formato del fichero es el siguiente:

1. En primer lugar, aparece una línea encabezada con el carácter # donde se describen las columnas del fichero (Latitud Longitud País Bandera)
2. A continuación, cada línea corresponde a los datos de un país concreto:
    - Valor de Latitud (-90º a 90º)
    - Valor de Longitud (-180º a 180º)
    - Nombre del país.
    - Nombre del fichero con la bandera del país.

2.3.3. Ficheros de imágenes en formato PGM

PGM es el acrónimo de *Portable GrayMap File Format*. Como hemos indicado anteriormente, el formato PGM es uno de los formatos que incorporan una cabecera. Un fichero PGM tiene, desde el punto de vista del programador, un formato mixto texto-binario: la cabecera se almacena en formato texto y la imagen en sí en formato binario. Las imágenes PGM almacene imágenes de niveles de gris (los valores de los pixeles están en el rango [0,255]. Con más detalle, la descripción del formato PGM es la siguiente:



1. **Cabecera.** La cabecera está en formato texto y consta de:
   - Un "número mágico" para identificar el tipo de fichero. Un fichero PGM que contiene una imagen digital de niveles de gris tiene asignado como identificador los dos caracteres P5.
   - Un número indeterminado de comentarios.
   - Número de columnas (c).
   - Número de filas (f).
   - Valor del mayor nivel de gris que puede tener la imagen (m). Cada uno de estos datos está terminado por un carácter separador (normalmente un salto de línea)
2. **Contenido de la imagen:** Una secuencia binaria de f ×c bytes, con valores entre 0 y m. Cada uno de estos valores representa el nivel de gris de un pixel. El primero referencia al píxel de la esquina superior izquierda, el segundo al que está a su derecha, etc.

Algunas aclaraciones respecto a este formato:

- El número de filas, columnas y mayor nivel de gris se especifican en modo texto, esto es, cada dígito viene en forma de carácter.
- Los comentarios son de línea completa y están precedidos por el carácter #. La longitud máxima es de 70 caracteres. Los programas que manipulan imágenes PGM ignoran estas líneas.
- Aunque el mayor nivel de gris sea m, no tiene porqué haber ningún pixel con este valor. Éste es un valor que usan los programas de visualización de imágenes PGM.

2.3.4. Ficheros de imágenes en formato PPM

PPM es el acrónimo de *Portable PixMap File Format*. . Este tipo de formato permite almacenar imágenes en color. Para cada pixel en este formato se alamcena el nivel de rojo, verde y azul, cada uno de ellos en el rango [0-255]. Al igual que las imágenes PGM las imágenes PPM estas compuestas de una cabecera y parte que describe el contenido de la imagen.

1. **Cabecera.** La cabecera está en formato texto y consta de:
    - Un "número mágico" para identificar el tipo de fichero. Un fichero PGM que contiene una imagen digital de niveles de gris tiene asignado como identificador los dos caracteres P6.
    - Un número indeterminado de comentarios.
    - Número de columnas (c).
    - Número de filas (f).
    - Valor del mayor nivel de gris que puede tener la imagen (m). Cada uno de estos datos está terminado por un carácter separador (normalmente un salto de línea)
2. **Contenido de la imagen:** Una secuencia binaria de f ×c tripletas de bytes, con valores entre 0 y m. Cada uno de esta trileta se corresponde con el valor rgb de un píxel. Así los tres primeros byte representa el nivel rgb del píxel de la esquina superior izquierda, la segunda triplete al que está a su derecha, etc.

3. Módulos

La implementación de los programas de esta práctica debe incluir, al menos, los siguientes módulos, que se probarán por separado y serán utilizados por los programas descritos en las secciones anteriores.

3.1. Módulo de imágenes

Para crear los distintos programas de esta práctica, será necesario desarrollar el T.D.A Imagen.

El módulo en el que se implemente el T.D.A. Imagen debe ser lo suficientemente flexible para poder implementar, sin demasiadas complicaciones, los tres programas propuestos en lasección anterior.

Una posible interfaz y representación del T.D.A Imagen podría ser la siguiente:

```
#ifndef __IMAGEN__H
#define __IMAGEN_H
struct Pixel{
        unsigned char r,g,b;
        unsigned char transparencia;
};
class Imagen{
 private:
        Pixel **datos; //donde se almacena la información de la imagen. Otra posible representación
                                //Pixel*datos
        int nf,nc;
 public:

        Imagen(); //contructor por defecto
         Imagen(const Imagen &I); //constructor de copia
        Imagen (int nf,int nc); //constructor con parámetros
        ~Imagen();
        Imagen & operator =(const Imagen & I);
        int getFilas()const; // devuelve el numero de filas nf
        int getColumnas()const; // devuelve el numero de columnas nc
        Pixel & operator()(int i,int j); // devuelve el píxel en la posición i,j
        const Pixel & operator()(int i,int j)const; // devuelve el píxel en la posición i,j
       void escribir (const char *nombre); // escribe en disco la imagen
       void leer(const char *nimagen, string nombre_mascara="");//Leer una imagen de disco.
….
};
#endif
```

Con respecto a la función *leer,* a ésta se le proporcionan dos parámetros. Uno es el nombre de la imagen en disco y el segundo es el nombre del fichero donde se encuentra la máscara asociada. Si la imagen no tiene máscara asociada, el parámetro nombre_mascara adopta el valor "" (cadena vacía). En este caso, cuando no hay una máscara asociada, se asume que todos los valores de la máscara corresponden a 255.

Cualquier otra función que se considere necesaria debe añadirse a la especificación anterior de la interfaz del T.D.A. Imagen. Por ejemplo, podría ser interesante añadirle a la clase Imagen un iterador que nos permita recorrer los píxeles de la imagen (o de una región de ésta).

3.2. Módulo de países

El T.D.A País representará un país, almacenando el punto geográfico asociado, el nombre del país y su bandera.

Así mismo, se debería desarrollar un T.D.A. Países que permita gestionar el conjunto de países que figuran en el fichero de países.

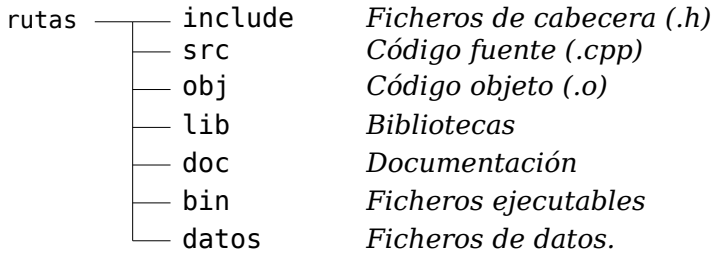
> NOTA: Obviamente, también harán falta los módulos Punto, Rutas y Almacén de Rutas desarrollados en la práctica 3 (STL e Iteradores).

4. Entrega de la práctica

Se deberán empaquetar todos los archivos relacionados con el proyecto en un archivo llamado "*rutas.tgz*", que habrá que entregar antes de la fecha que se publicará en la página web de la asignatura. No incluya incluirán ficheros objeto, ni ejecutables, ni ficheros de datos.

Se incluir el archivo *Makefile* para realizar la compilación y conviene utilizar este *Makefile* para asegurarse de que se eliminan todos los archivos temporales e intermedios que se puedan generar a partir de los ficheros con el código fuente (con make clean).

Los archivos del proyecto deben estar distribuidos en directorios de la siguiente forma*:*



Para realizar la entrega de la práctica, asegúrese de eliminar los archivos que no se incluirán en ella, sitúese en el directorio que incluya la carpeta de su proyecto y ejecute:

```
tar zcv rutas.tgz rutas
```

Este comando creará un nuevo archivo *rutas.tgz* que contendrá el directorio del proyecto completo, con todos los subdirectorios y y archivos que cuelguen de él.

5. Referencias

[GAR06b] Garrido, A. & Fdez-Valdivia, J. "*Abstracción y estructuras de datos en C++*". Delta publicaciones, 2006.

[ED2013] Práctica 3: STL e Iteradores. Estructuras de Datos, curso 2013/2014.