UNIVERSIDAD DE GRANADA

Departamento de Ciencias de la Computación e Inteligencia Artificial

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

Grado en Ingeniería Informática
Doble Grado en Ingeniería Informática y Matemáticas
Doble Grado en Ingeniería Informática y ADE

1.- Introducción y Objetivo

El objetivo de esta práctica es 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.
6. Haber realizado la Práctica 2: TDA Imagen. En el caso que el estudiante no haya trabajado esta práctica es recomendable entenderla.

El objetivo de esta práctica es llevar a cabo el análisis, diseño e implementación de un proyecto. Con tal fin el estudiante 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.

2.- Descripción de la Práctica

Se desea crear un software para ayudar a un compañía aérea poder visualizar sus rutas (por qué países pasa) . En concreto la empresa aérea querrá visualizar en un mapa del mundo rutas con un código determinado, p.ej. R2. Nuestro programa debería obtener una imagen y al mismo tiempo la secuencia de países que atraviesa. (ver figura 1). En pantalla deberá aparecer: Canadá Estados Unidos Perú España China Australia Japón Rusia

Figura 1.- Visualización de una ruta sobre el mapa. En este caso la ruta es R2: Canadá Estados Unidos Perú España China Australia Japón Rusia

Para poder visualizar la ruta aérea se pintarán las banderas de los países por los que pasa y además se marcará la ruta con secuencias de aviones que se orientan según la línea que une dos países.

2.1 Tareas a realizar

El alumno debe llevar a cabo las siguientes tareas:

1. Dar una especificación de los T.D.A necesarios. P.ej:
    - Punto
    - Ruta
    - Almacén de Rutas.
2. Definir el conjunto de operaciones con su especificación
3. Probar cada función con programas test antes de integrarla en su módulo correspondiente.

Se puede hacer uso de la STL.

A continuación se detallan los programas que el estudiante deberá desarrollar.

2.1.1 Rotar una imagen

Esta función permitirá al usuario dada una imagen y un ángulo obtener una nueva imagen que es la versión rotada, en el numero de grados dado, de la primera imagen. Este programa deberá, para probarlo, llamarse desde la línea de órdenes de la siguiente manera:

```
prompt% pruebarotacion avion.ppm 45 avion_45.ppm
```

en esta llamada se indica la imagen original “avion.ppm” el ángulo de rotacion 45 y el nombre de la imagen de salida. Ambas imágenes deben estar en formato ppm. Ver en detalle el formato ppm (ver secciones 2.2.3 y 2.3.4).

Para realizar la rotación el estudiante tiene a su disposición el código que se encuentra en el fichero pruebarotacion.cpp y el modulo imagenES para realizar la lectura y escritura de imágenes PPM. Simplemente para construir este programa deberá generar el módulo Imagen.

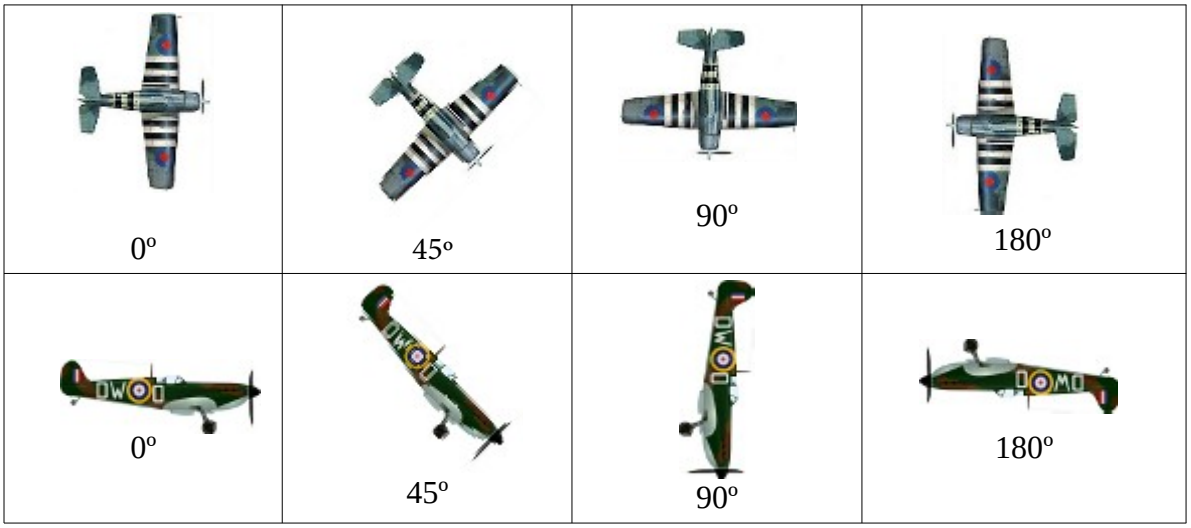


Figura 2: Una imagen rotada con diferentes ángulos

Nota: En el material asociado a la práctica tenéis el fichero pruebarotación.cpp donde tenéis 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 origen.

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

2.1.2 Pegado de una imagen en otra

Esta función permitirá al usuario, dada una imagen de fondo y una segunda imagen superponer la segunda imagen sobre la imagen de fondo. Para probar la función, el comando que se deberá ejecutar en la línea de órdenes será de este tipo:

```
prompt%  pruebapegado espana_reshigh.ppm avion2.ppm mascara_avion2.pgm
espana_avion2_blending.ppm 0 0 1
```

Los **parámetros** 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 (formato ppm)
3. Nombre de la imagen máscara de la segunda imagen.
4. Nombre de la imagen de salida.
5. Fila y Columna a partir de la cual empieza el pegado.
6. Un valor *0* si el pegado es opaco (no se ve nada de la imagen de fondo) o *1* si se hace un pegado tipo “blending” (se hace promedio entre la imagen fondo y la imagen a pegar).

Los resultados obtenidos ejecutando el ejemplo dando los diferentes posibles valores para la opacidad (0 o 1) se pueden ver en la figura 3

Imagen de fondo. If(i,j)

Imagen a pegar . I2(i,j)

Máscara de la imagen a pegar. M(i,j)

Pegado Opaco. Ir(i,j)

Pegado Blending. Ir(i,j)

Figura 3: Resultados del pegado de una imagen sobre otra. A la izquierda abajo pegado opaco. A la derecha abajo pegado con transparencia

El proceso de pegado sigue los siguientes pasos:

1. Sea If la imagen de fondo.
2. Sea Ir la imagen resultante
3. Sea I2 la imagen a pegar
4. posi y posj la fila y columna donde se comienza a pegar.
5. 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 máscara es una imagen con dos niveles de gris 0 y 255. Un valor 0 nos indica que ese píxel no será tenido en cuenta en el proceso de pegado y un valor 255 sí. Estas imágenes se almacena en formato PGM (ver secciones 2.2.3 y 2.3.4 para mayor detalle).

2.1.3 Ruta Aérea

Este programa obtiene un mapa del mundo sobre el que se ha pintado una ruta elegida. Para tal objetivo el usuario insertará un fichero con un conjunto de países (ver especificación del fichero paises en la sección 2.3.2) y un almacén de rutas y una imagen de un mapa del mundo junto con la imagen del avión mostrada en la figura 2 y su máscara. Un ejemplo del resultado deseado es el que se muestra en figura 1. Además se debe indicar el directorio donde se almacena las banderas de cada país (en el ejemplo siguiente dir_banderas). Para probar esta función, deberá llamarse desde la línea de órdenes de la siguiente manera:

```
prompt% ruta_aerea  paises.txt mapa.ppm dir_banderas almacen_rutas.txt
avion.ppm mascara_avion.pgm
```

El programa en primer lugar mostrará en la consola todas las rutas existentes en el almacén de rutas y pedirá al usuario que inserte el código de una ruta. Tras esto el programa almacenará en disco una

Las rutas:
R1 5 (34.5204,69.2008) (52.5079,13.4261) (7.40665,12.3446) (-0.186596,-78.4305) (40.4005,-3.59165)
R2 8 (58.6954,-96) (35.0869,-103.723) (-12.0553,-77.0452) (40.4005,-3.59165) (37.9438,104.136) (-27.7871,133.281) (35.6735,139.71) (62.8865,61.5512)
R3 5 (17.2464,-19.6706) (4.28364,-74.224) (51.5289,-0.101599) (62.8865,61.5512) (37.9438,104.136)
R4 11 (14.4225,-87.6343) (48.8589,2.34706) (24.7259,46.8225) (58.6954,-96)(35.0869,-103.723) (-12.0553,-77.0452) (40.4005,-3.59165) (37.9438,104.136) (-27.7871,133.281) (35.6735,139.71) (62.8865,61.5512)
R5 5 (52.7608,8.74761) (-19.0519,29.1528) (-34.6159,-58.4333) (58.6954,-96)(52.7608,8.74761)
R4
Honduras Francia ArabiaSaudita Canada EstadosUnidos Peru España China Australia Japon Rusia

imagen ppm que cuyo nombre es el código de la ruta escogida y también mostrará en la consola la secuencia de países por los que pasa la ruta. En la imagen aparece en la localización del país por donde pasa la bandera del país. Además entre cada dos puntos de la ruta se pintará el avión al comienzo, punto intermedio y final.

La traslación de un par (latitud,longitud) a un (i,j)-pixel de la imagen se hace mediante la siguiente ecuación:

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

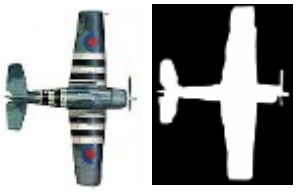
Siendo *totalcolumnas* y *totalfilas* el numero de filas y numero de columnas de la imagen con el mapa.

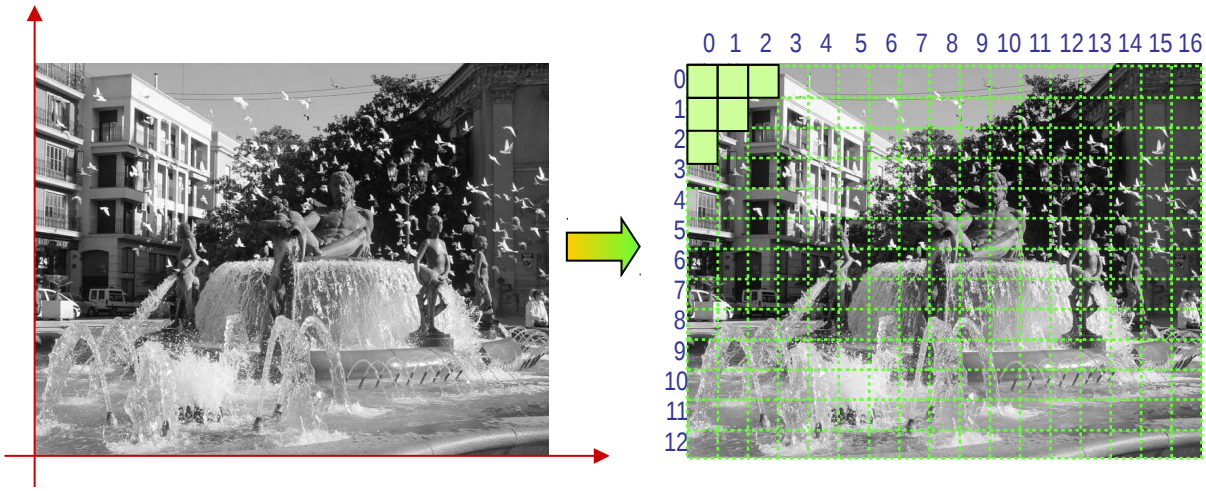
2.2 Imágenes

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.

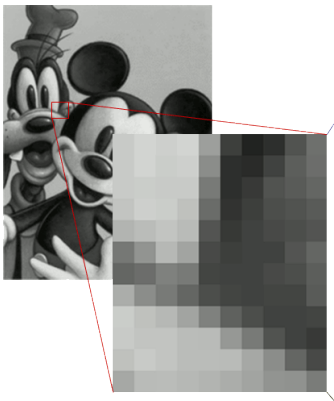




2.2.1 Imágenes en blanco y negro.

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

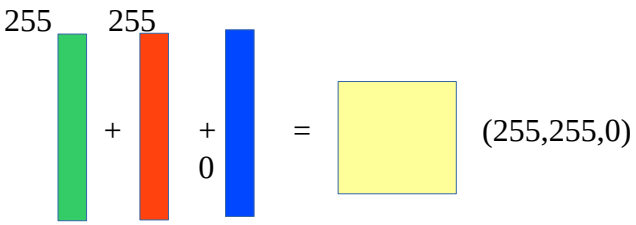
2.2.2 Imágenes en color

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 entrada 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 separadores.
    - El numero de puntos de la ruta. A continuación los puntos geográficos separados por “blancos” (espacios, tabuladores,..). Cada punto se especifica con dos valores reales que especifican la latitud y longitud. Estos valores se especifican entre paréntesis separados por una coma.
3. A continuación pueden aparecer o no puntos de interés. Si se especifica aparecerá la palabra en un línea #Puntos_de_Interes. A continuación en las siguientes líneas se especifican los puntos de interés con su descripción.

2.3.2 Países

Un ejemplo de fichero de países es el que se muestra a continuación:

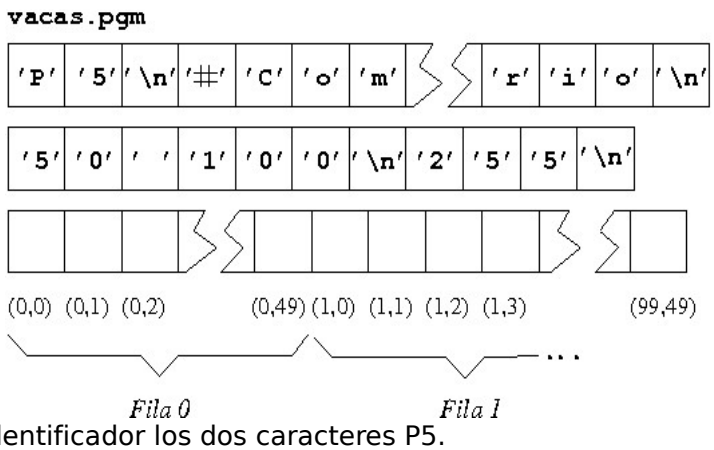
```
# Latitud              Longitud              Pais           Bandera
34.520418555522845     69.20082090000005     Afganistan     afganistan.ppm
41.332136072796175     19.812877200000003    Albania        albania.ppm
52.50786264022465      13.426141949999987    Alemania       alemania.ppm
-11.294616098942507    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 describe las columnas del fichero (Latitud Longitud País Bandera)
2. A continuación cada línea se corresponde con la información de un país:
    - Valor de Latitud (-90º a 90º)
    - Valor de Longitud (-180º a 180º)
    - El nombre del país.
    - El fichero con la bandera del país.

2.3.3 Ficheros de Imagen 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 Imagen 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 a desarrollar.

Módulo Imagen

Para crear los programas propuestos hará falta desarrollar el T.D.A Imagen. Este módulo debe ser lo suficiente flexible para poder desarrollar los tres programas propuestos. Para ello 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 GetCols()const; //devuelve el numero de filas nc
        Pixel & operator()(int i,int j); // devuelve el pixel en la posicion i,j
        const Pixel & operator()(int i,int j)const; // devuelve el pixel en la posicion 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 esta se le proporcionan dos parámetros. Uno el nombre de la imagen en disco. Y el segundo es el nombre del fichero donde se encuentra la imagen máscara asociada. Si no tiene imagen máscara asociada el parámetro nombre_mascara adopta el valor “”. En este caso sin máscara asociada todos los valores de la transparencia de cada píxel se pone a 255.

Cualquier otra función que veáis necesaria deberíais añadirla. También sería interesante añadirle a la clase Imagen un **iterador** que nos permita recorrer los píxeles de la imagen.

Módulos País y Países

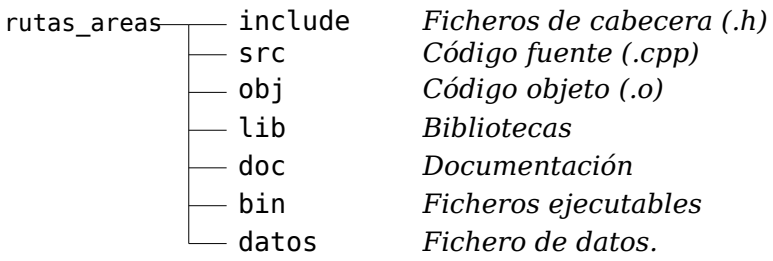
El T.D.A Pais representará un país, almacenando el punto geográfico asociado, el nombre del país y su bandera. Por otro lado deberías almacenar todo el conjunto de Países que se proporcionan en el fichero países. Para ellos deberéis desarrollar el T.D.A Países.

Para la práctica hará falta también implementar las **clases Punto, Rutas y Almacén de Rutas**.

4.- Práctica a entregar

1.- El estudiante deberá empaquetar todos los archivos relacionados en el proyecto en un archivo con nombre "*rutas_areas.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 estudiante 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 "*rutas_aereas*") para ejecutar:

```
prompt% tar zcv rutas_aereas.tgz rutas_aereas
```

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

2.- La práctica se puede realizar ***por parejas***

3.- La puntuación máxima de la práctica es de ***1.25 puntos***

5.- Referencias

- [GAR06b] Garrido, A. Fdez-Valdivia, J. "*Abstracción y estructuras de datos en C++*". Delta publicaciones, 2006.
- [ED2013]. Práctica 2: TDA Imagen. Curso 2020-2021 de Estructuras de Datos.