PRÁCTICA 2 – Compresión de imágenes

El objetivo es escribir un programa que sea capaz de comprimir y descomprimir un determinado tipo de imágenes creadas especialmente para la práctica.
Primero vamos a ver como será el formato de las imágenes a comprimir/descomprimir y el método que utilizaremos. Posteriormente se detallará el funcionamiento de la práctica.

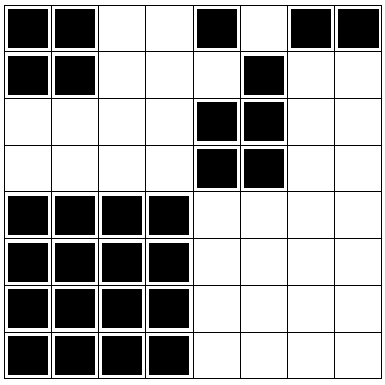
Formato de las imágenes

Las imágenes que utilizaremos serán en blanco y negro, almacenadas en ficheros de texto y con un tamaño muy concreto. La estructura de un fichero de imagen será la siguiente:

- La primera línea indicará el tamaño de la imagen. Será siempre un valor potencia de 2 que pueda almacenarse en una variable de tipo `word`. Tendrán igual número de filas y de columnas.
- Las siguientes líneas contendrán la imagen propiamente dicha. Cada línea del fichero de texto representa una fila de píxeles de la imagen, y cada carácter de la línea representa un píxel.
  - Un carácter 0 indica color BLANCO.
  - Un carácter 1 indica color NEGRO.

Un ejemplo de fichero de imagen sería el siguiente representando la imagen que está a su lado (será la que utilicemos como ejemplo de ahora en adelante):

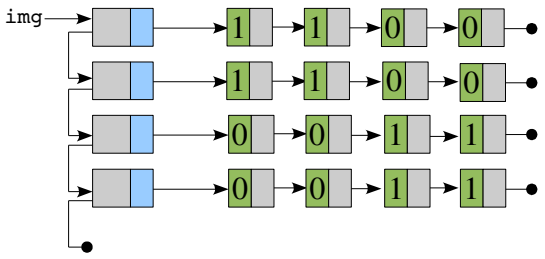
```
8
11001011
11000100
00001100
00001100
11110000
11110000
11110000
11110000
```



Representación de las imágenes en memoria

Vamos a utilizar una estructura de datos especial para almacenar una imagen en memoria interna. Ha de quedar claro que este método no es una forma común de guardar imágenes en memoria ya que no es eficiente, pero es el que se debe utilizar en nuestro caso.

Se guardarán las imágenes en memoria utilizando listas enlazadas. Una imagen será un puntero a una lista enlazada de filas, donde cada nodo que representa una fila será una lista enlaza de píxeles.
El esquema quedaría como se ve en la figura para una imagen de 4x4.



Método de compresión/descompresión

Para comprimir las imágenes nos basaremos en un principio de compresión de imágenes que dice que si se selecciona un píxel aleatorio de la imagen, existe una probabilidad bastante alta de que los píxeles vecinos (los que están a su alrededor) tengan el mismo (o muy similar) color que dicho píxel (en nuestro caso o es el mismo color o no, ya que solo usamos dos colores).

Para hacer uso del principio anterior, utilizaremos una conocida estructura de datos llamada **quadtree**. Un quadtree es un árbol cuyos nodos, o son hojas, o tienen cuatro hijos. Solamente se pueden dar esos dos casos. Nunca hay nodos no hojas con un número de hijos diferente a cuatro. Cada nodo hoja almacenará un valor correspondiente a un color (blanco o negro).

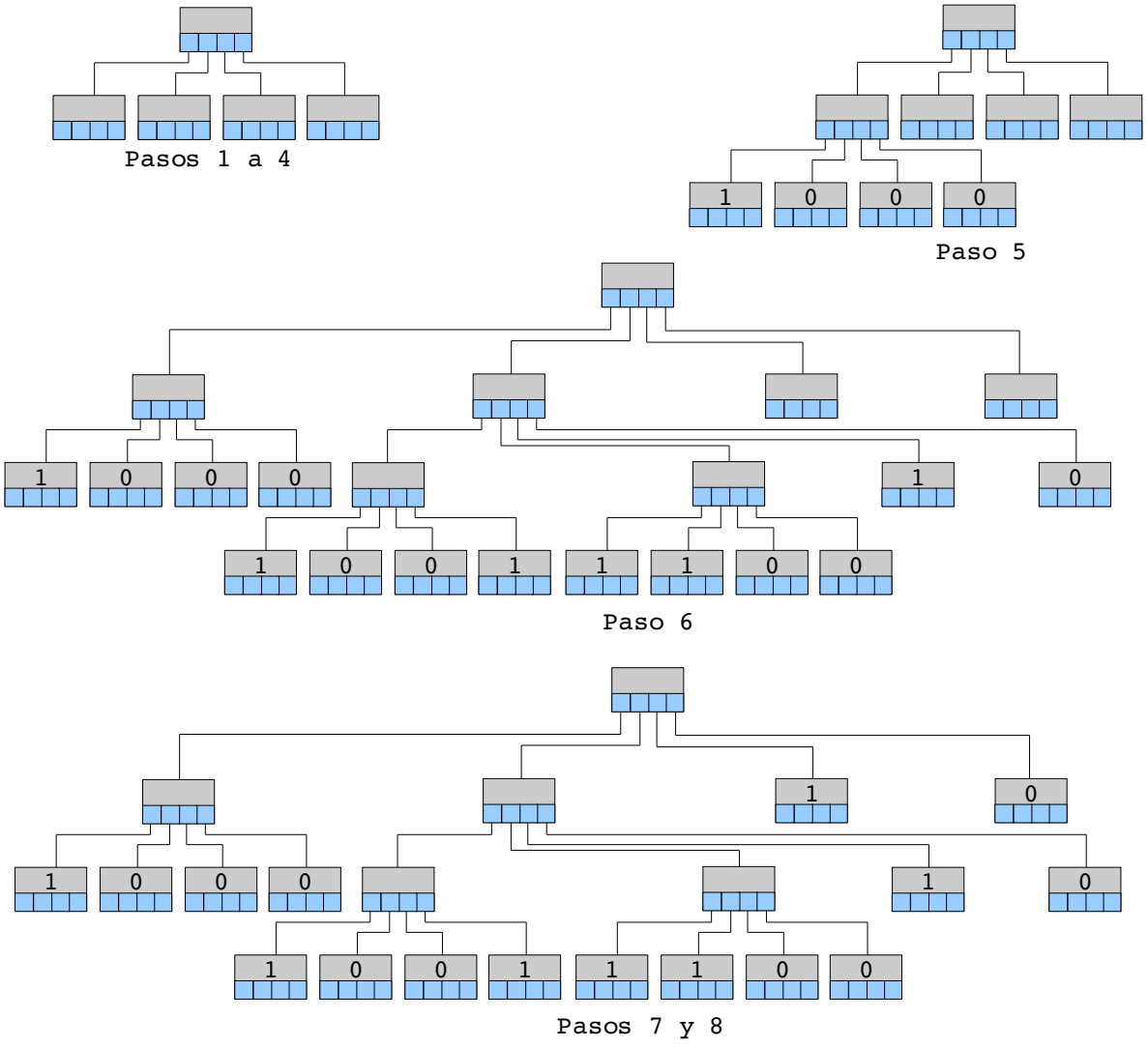
El método comienza creando el nodo raíz del árbol. Se examina la imagen. Si todos los píxeles de la imagen son iguales (la imagen es uniforme), entonces con el nodo raíz que se ha creado podemos representar toda la imagen.
Si en la imagen existen píxeles de diferente color, entonces dividimos la imagen en cuatro cuadrantes del mismo tamaño, cada uno de los cuales se convierte en un nuevo nodo del árbol, hijo del nodo raíz. Un cuadrante uniforme (el que tiene todos los píxeles iguales) se guarda como nodo hoja en el árbol, mientras que un cuadrante no uniforme (el que tiene píxeles diferentes) se guarda como un nodo interno. Los cuadrantes no uniformes se dividen recursivamente en subcuadrantes más pequeños que se guardan como cuatro nodos hermanos en el quadtree.
Numeraremos los cuadrantes del 1 al 4, correspondiendo el 1 al superior-izquierda, el 2 al superior-derecha, el 3 al inferior-izquierda y el 4 al inferior-derecha. Siempre que se realiza una subdivisión, los cuadrantes resultantes se numerarán de esa forma.

Aquí se muestran los pasos para formar el quadtree correspondiente a la imagen anterior:

```
1 - Creamos nodo raíz.
2 - Examinamos la imagen. Como no es uniforme el nodo raíz no es una hoja.
3 - Dividimos la imagen en cuatro cuadrantes del mismo tamaño.
4 - Creamos cuatro nodos hijos del raíz. Cada uno corresponderá a un cuadrante.

5 - El cuadrante 1 NO es uniforme, por lo que no puede ser nodo hoja.
    5.1 - Dividimos el cuadrante 1 anterior en cuatro nuevos cuadrantes y creamos cuatro
      nodos hijos para cada uno de ellos. De los cuatro cuadrantes resultantes tenemos:
       5.1.1 - Los cuatro cuadrantes son uniformes, el 1 es negro y los otros tres son
         blancos. Cada uno de los cuatro nodos creados serán nodos hoja, el primero con
         valor negro asociado y los otros tres con valor blanco.

6 - El cuadrante 2 NO es uniforme, por lo que no puede ser nodo hoja.
    6.1 - Dividimos el cuadrante 2 anterior en cuatro nuevos cuadrantes y creamos cuatro
      nodos hijos para cada uno de ellos. De los cuatro cuadrantes resultantes tenemos:
       6.1.1 - El cuadrante 1 NO es uniforme, por lo que no puede ser nodo hoja.
          6.1.1.1 - Dividimos el cuadrante 1 anterior en cuatro nuevos cuadrantes y
            creamos cuatro nodos hijos para cada uno de ellos. De los cuatro cuadrantes
            resultantes tenemos:
             6.1.1.1.1 - Hemos llegado al nivel mínimo (un cuadrante es un píxel), por
               lo que los cuadrantes son uniformes. Cada uno de los cuatro nodos será
               una hoja. El 1 será negro, el 2 será blanco, el 3 será blanco y el 4
               será negro.
       6.1.2 - El cuadrante 2 NO es uniforme, por lo que no puede ser nodo hoja.
          6.1.2.1 - Dividimos el cuadrante 2 anterior en cuatro nuevos cuadrantes y
            creamos cuatro nodos hijos para cada uno de ellos. De los cuatro cuadrantes
            resultantes tenemos:
```

```
            6.1.2.1.1 - Hemos llegado al nivel mínimo (un cuadrante es un píxel), por
              lo que los cuadrantes son uniformes. Cada uno de los cuatro nodos será
              una hoja. El 1 será negro, el 2 será negro, el 3 será blanco y el 4 será
              blanco.
        6.1.3 - El cuadrante 3 SI es uniforme, de color negro, por lo que será un nodo
          hoja con valor negro asociado.
        6.1.4 - El cuadrante 4 SI es uniforme, de color blanco, por lo que será un nodo
          hoja con valor negro asociado.

7 - El cuadrante 3 SI es uniforme, de color negro, por lo que será un nodo hoja con
  valor negro asociado.

8 - El cuadrante 4 SI es uniforme, de color blanco, por lo que será un nodo hoja con
  valor blanco asociado.
```

Aquí se puede ver la evolución del árbol según los pasos anteriores:



Una vez que tenemos el árbol en memoria, el paso contrario, el de descompresión, puede realizarse sin problemas ya que en todo momento conocemos el tamaño de la imagen. Al recorrer el árbol podremos asociar a cada nodo su correspondiente cuadrante. Si el nodo es una hoja, pintaremos dicho cuadrante del color que indique el nodo, y sino, visitaremos sus hijos para pintar sus subcuadrantes de forma recursiva.

Almacenar el árbol en disco

Una vez generado el árbol que representa la imagen comprimida, hay que almacenarlo en un fichero para luego poder cargarlo y volver a obtener la imagen original. Será un fichero de bytes y su estructura será tal y como se describe a continuación.

Los dos primeros bytes se reservan para almacenar la dimensión de la imagen. Si ésta cabe en un byte, el primer byte del fichero almacenará valor cero y el segundo dicha dimensión. Si no cabe en un byte, convertir el tamaño en decimal a un número binario de 16 bits y luego tomar cada grupo de 8 bits y guardarlos en los dos primeros bytes del fichero (por ejemplo, si la dimensión es 1024, lo pasáis a binario, que sería 0000010000000000; así el primer byte del fichero sería 4 y el segundo 0).

Una vez almacenado el tamaño, se almacenarán los nodos del árbol. Para que la interpretación al leer desde el fichero no sea ambigua, hemos de guardar los nodos de una forma muy concreta. Los nodos tendrán los siguientes valores:

```
Nodo interno (no hoja)  ->    valor 0
Nodo hoja blanco        ->    valor ord('0') = 48
Nodo hoja negro         ->    valor ord('1') = 49
```

Se guardarán los nodos en el orden resultante de realizar un recorrido en anchura del árbol. Se guardará el raíz, posteriormente los valores correspondientes a los cuatro hijos del raíz, luego todos los hijos resultantes de esos cuatro hijos, etc.

Aquí se muestra el fichero resultante de guardar el árbol generado en el ejemplo anterior:

```
Byte  1 --> 0
Byte  2 --> 8 (en binario, 00001000)
Byte  3 --> 0 (nodo raíz)
Byte  4 --> 0 (hijo 1 del raíz; nodo interno)
Byte  5 --> 0 (hijo 2 del raíz; nodo interno)
Byte  6 --> 49 (hijo 3 del raíz; valor negro = '1')
Byte  7 --> 48 (hijo 4 del raíz; valor blanco = '0')
Byte  8 --> 49 (hijo 1 del nivel de profundidad 3; valor negro = '1')
Byte  9 --> 48 (hijo 2 del nivel de profundidad 3; valor blanco = '0')
Byte 10 --> 48 (hijo 3 del nivel de profundidad 3; valor blanco = '0')
Byte 11 --> 48 (hijo 4 del nivel de profundidad 3; valor blanco = '0')
Byte 12 --> 0 (hijo 5 del nivel de profundidad 3; nodo interno)
Byte 13 --> 0 (hijo 6 del nivel de profundidad 3; nodo interno)
Byte 14 --> 49 (hijo 7 del nivel de profundidad 3; valor negro = '1')
Byte 15 --> 48 (hijo 8 del nivel de profundidad 3; valor blanco = '0')
Byte 16 --> 49 (hijo 1 del nivel de profundidad 4; valor negro = '1')
Byte 17 --> 48 (hijo 2 del nivel de profundidad 4; valor blanco = '0')
Byte 18 --> 48 (hijo 3 del nivel de profundidad 4; valor blanco = '0')
Byte 19 --> 49 (hijo 4 del nivel de profundidad 4; valor negro = '1')
Byte 20 --> 49 (hijo 5 del nivel de profundidad 4; valor negro = '1')
Byte 21 --> 49 (hijo 6 del nivel de profundidad 4; valor negro = '1')
Byte 22 --> 48 (hijo 7 del nivel de profundidad 4; valor blanco = '0')
Byte 23 --> 48 (hijo 8 del nivel de profundidad 4; valor blanco = '0')
```

Obtener el árbol en memoria a partir del fichero solamente requiere ir recorriendo el fichero secuencialmente, y construir el árbol en anchura (y no en profundidad, que fue como se construyó a partir de la imagen).

Pequeñas ayudas

Aquí os pongo algunas pistas para que tengáis una base sobre la que trabajar.

– Esquema del procedimiento que genera el árbol a partir de una imagen:

```
procedimiento generarArbol(imagen, arbol, coordenadas);
begin
      para cada cuadrante c hacer
      begin
            calcular las nuevas coordenadas de c;
            crear nuevo nodo nc que lo representa;
      end;

      para cada cuadrante c hacer
      begin
            si c no es uniforme entonces generarArbol(imagen,nc,c);
            sino poner valor de nc al color del cuadrante c;
      end;
end;
```

– ¿Cómo calcular las nuevas coordenadas de un determinado cuadrante?

Dado un determinado cuadrante `c` definido por `(x1,y1) , (x2,y2)`, con la numeración de cuadrantes fijada al inicio del documento, tenemos que las coordenadas de los cuatro subcuadrantes de `c` (`sc1, sc2, sc3 y sc4`) son:

`n = ⌊(c.x2-c.x1)/2⌋`

```
sc1.x1 = c.x1          sc2.x1 = c.x1          sc3.x1 = c.x1 + n + 1 sc4.x1 = c.x1 + n + 1
sc1.y1 = c.y1          sc2.y1 = c.y1 + n + 1 sc3.y1 = c.y1         sc4.y1 = c.y1 + n + 1
sc1.x2 = c.x1 + n      sc2.x2 = c.x1 + n      sc3.x2 = c.x2         sc4.x2 = c.x2
sc1.y2 = c.y1 + n      sc2.y2 = c.y2          sc3.y2 = c.y1 + n     sc4.y2 = c.y2
```

– Esquema del procedimiento para guardar el árbol en el fichero

```
procedimiento guardarArbol( ... );
begin
      guardar nodo raiz;
      crear una cola;
      encolar el nodo raiz;
      mientras la cola no este vacia
      begin
            sacar nodo de la cola;
            si no es hoja entonces
            para cada nodo hijo
            begin
                  guardar valor en el fichero;
                  encolar el nodo;
            end;
      end; {while}
end;
```

Estas pistas también os deberían servir para los procedimientos de cargar el árbol desde el fichero (también se resuelve cómodamente con una cola) y para el que forma la imagen a partir del árbol (también requiere del cálculo de los cuadrantes y ha de ser recursivo).

Funcionamiento de la práctica

La práctica debe permitir comprimir imágenes introducidas desde la entrada estándar o desde un fichero y debe permitir descomprimir ficheros previamente comprimidos. El programa funcionará según las opciones que se le pasen por la línea de comandos. Para gestionar estas opciones utilizaréis la librería `GetOpts`.

Las opciones a gestionar son[1]:

- `-c`: esta opción no lleva argumentos. Indica que el programa tiene que comprimir. Si no aparece esta opción es obligatorio que aparezca la opción `-e`.
- `-e`: esta opción no lleva argumentos. Indica que el programa tiene que descomprimir. Si no aparece esta opción es obligatorio que aparezca la opción `-c`.
- `-f nombre_fichero`: establece el fichero de entrada. Esta opción no es obligatoria. Estos son los casos que se pueden producir:
  - Si se especifica junto con la opción `-c`, entonces el fichero especificado tiene que ser un fichero de texto, y la imagen a comprimir se carga desde dicho fichero, no permitiendo introducir texto por la entrada estándar.
  - Si se especifica junto con la opción `-e`, entonces el fichero especificado tiene que ser un fichero binario comprimido.
  - Si no se especifica esta opción el programa recibirá la imagen a comprimir por la entrada estándar con formato idéntico al de los ficheros de imágenes. En este caso el programa funcionará como un filtro de unix, es decir, ni pide nada ni escribe nada a la salida estándar, simplemente lee de la entrada estándar interpretando ésta como si fuera un fichero imagen. (Ver variable `Input` de la unidad `System`).

    Se deduce de todos los casos que la opción `-e` no puede darse sin la opción `-f` (siempre que se vaya a descomprimir hay que especificar el fichero a descomprimir).
- `--output="nombre_fichero"`: esta es una opción larga y no es obligatoria. Dos casos:
  - Se pone junto con la opción `-c`. Entonces sirve para especificar el nombre del fichero comprimido de salida. Si no se usa esta opción cuando comprimimos, entonces el nombre del fichero de salida será `imagencomp.dat`.
  - Se pone junto con la opción `-e`. Entonces sirve para especificar el nombre del fichero de texto donde se almacenará el texto obtenido al descomprimir. Si no se usa esta opción cuando descomprimimos, entonces la imagen descomprimida se mostrará en un fichero llamado `imagenfinal.txt`.
- `--verbose`: esta es una opción larga y no es obligatoria. Si se pone, entonces el programa tiene que mostrar la siguiente estadística por pantalla:
  - Dimensión de la imagen.
  - Número de píxeles blancos.
  - Número de píxeles negros.
  - Ratio de compresión. Será el cociente de dividir el número de píxeles de la imagen completa por el número de nodos del árbol que se ha generado. Se mostrará redondeado con dos decimales.

1 Es obligatorio que el programa implemente TODAS las opciones que figuran aquí. En los puntos donde se menciona que una opción no es obligatoria, significa que es opcional ponerlo en la línea de comandos al llamar al programa, pero si se pone, el programa la tiene que procesar.

Observaciones

- El tamaño mínimo de imagen será de 2x2.
- El TAD `quadtree` (definición de tipos y operaciones) debe estar en una unidad separada.
- El TAD `imagen` (definición de tipos y operaciones) debe estar en una unidad separada.
- El procedimiento que genera el árbol a partir de la imagen tiene que ser recursivo.
- El procedimiento que forma la imagen a partir del árbol tiene que ser recursivo.
- Es obligatorio hacer control de errores de entrada/salida. No se dará apta ninguna práctica que de fallos en tiempo de ejecución por motivos de trabajo con ficheros.
- Se prohíbe el uso de las funciones: `Delay, GotoXY, WhereX, WhereY, TextColor, TextBackground, Sound, NoSound`. También se prohíbe usar la unit `Graph`.
- Es obligatorio que el código esté identado y comentado. Para cada función y procedimiento han de figurar los siguientes datos en los comentarios:
  - Descripción del valor de retorno (en caso de funciones).
  - Descripción de los parámetros de la función o procedimiento (si son de entrada o salida y para qué se utilizan).
  - Pequeña descripción de lo que hace la función o el procedimiento.

Normas de entrega

- El programa debe implementarse utilizando el compilador Free Pascal versión 2.0.2 (no sirve otra versión) en plataforma Linux. Para la corrección se compilará el código fuente y posteriormente se ejecutará el programa. Dicha compilación/ejecución se realizará en una máquina con Linux y el compilador Free Pascal. Si el alumno utilizara otro compilador o sistema operativo para desarrollar el programa es responsabilidad suya el comprobar que funciona bajo las condiciones mencionadas.
- Se puede realizar la práctica individualmente o en grupos de dos personas. En caso de hacerlo en grupo se entregará una práctica por grupo.
- La entrega de las prácticas ha de seguir las siguientes normas:
  - Hay que entregar código fuente y ejecutable para sistemas Linux.
  - Al hacer login con vuestros datos de alta en la asignatura, encontraréis en vuestra carpeta de usuario una carpeta llamada “EntregaPracticaImagenes” (no hay que crearla). Debéis dejar ahí los ficheros antes de la fecha indicada ya que posteriormente no podréis escribir en ella.
  - Hay que enviar un correo electrónico a la dirección carlos.testera@unileon.es sin contenido en el cuerpo del mensaje (y sin enviar la práctica como adjunto).
    El asunto del correo tiene que ser de la forma: << Práctica imágenes – [Iniciales – dni] >>
    Ej.--> Práctica imágenes – [Abc – 71223344Z]
    Si la práctica la entregan dos personas se pondrán los datos seguidos. Un ejemplo sería:
    Ej. --> Práctica imágenes – [Abc – 71223344Z][Hjk – 71998877A]
    (IMPORTANTE: Es fundamental que respetéis el formato del asunto tal y como se muestra, ya que se usará un filtro para desviar los mensajes que contienen las prácticas. Los que no cumplan dicho formato no serán detectados y no se tendrán en cuenta).

La fecha límite de entrega es el 31 de Mayo de 2007