Guion III: Compresión sin pérdida Codificación Aritmética

Información sobre la entrega de la práctica

Las prácticas se entregarán en un único fichero comprimido Practica03ApellidoNombre.zip. El fichero c+ontendrá:

- Las funciones de Matlab a realizar en ficheros .m con los nombres de las funciones que se indiquen en el guion.
- Los trozos de código a realizar, que se entregarán todos en los pasos correspondientes de un único fichero .m llamado Practica03ApellidoNombre.m . Este fichero lo crearás modificando el fichero .m Practica03MolinaRafael.m en el servidor.
- Las discusiones y respuestas solicitadas en el guion se entregarán en un único fichero pdf. El nombre del fichero será Practica03ApellidoNombre.pdf. Lo construirás editando Practica03MolinaRafael.doc y salvándolo en formato pdf.

Codificación aritmética

En este apartado estudiaremos la codificación aritmética, un método que, al igual que la codificación Huffman, pretende reducir el número medio de bits requeridos para representar un símbolo pero que, a diferencia de ésta, permite representar símbolos con un número de bits fraccionario. Observa que para que el número de bits correspondientes a un símbolo sea fraccionario utilizando Huffman necesitamos codificar conjuntamente pares, tripletas, etc de símbolos, con los problemas de requerimiento adicional de espacio que esto genera.

Además del material que veremos en esta práctica te recomendamos que visites la codificación aritmética de Michael Dipperstein. Puedes usar el código en C de codificación aritmética de Michael Dipperstein para estudiar el funcionamiento práctico de la codificación aritmética mediante la realización de una serie de ejercicios o el código de las correspodientes funciones de Matlab que ahora veremos.

Por último antes de realizar algunos ejemplos observa que para saber cuándo hemos terminado de decodificar todos los datos podemos, bien podemos indicar el número de elementos que vamos a codificar o introducir un símbolo adicional (**EOF**) que sólo aparece una sola vez y que se incluye en la codificación de la secuencia. En la decodificación de la secuencia, la aparición de este símbolo indica que hemos terminado.

Visita también la página web del curso sobre Multimedia de David Marshall de la Universidad de Bristol http://www.cs.cf.ac.uk/Dave/Multimedia/ . En él encontrarás, en la sección Online Course Notes, pdfs de temas interesantes relacionados con el curso. En la sección BSc Multimedia (CM3106) Tutorial/Lab Class Notes, Exercises, example Code and Libraries encontrarás tutoriales y código en Matlab. Mira en particular la clase 7. Una copia local de Basic_compression.zip, que contiene ficheros .m de Matlab, la puedes encontrar en el material de la asignatura.

Paso 1

Vamos a comenzar con un ejemplo de Matlab. Consideremos una fuente con alfabeto {x, y, z}. La fuente ha generado la secuencia yzxxx que queremos codificar. Declaramos el alfabeto y la secuencia observada

```
close all; clear all; clc;
alf=['x' 'y' 'z'];
seqob=['y' 'z' 'x' 'z' 'z'];
```

Paso 2

Convertimos la secuencia observada a la secuencia de los índices correspondientes de la matriz del alfabeto. Los índices van del *1*, que corresponde al primer símbolo del alfabeto, a la *longitud del alfabeto*, es decir numel(alf), que corresponde al último símbolo del alfabeto

```
indseqob =[2 3 1 3 3];
```

Paso 3

Vamos a crear la secuencia de índices observados mediante órdenes de Matlab y no manualmente como hemos hecho en el paso 2. Inicializa la matriz que contendrá dichos índices (más tarde entenderás porque creamos la matriz de tipo uint16)

```
indseqob=zeros(1,numel(seqob),'uint16');
```

Paso 4

Incluye en el paso 4 del fichero Practica03ApellidoNombre.m el código para convertir la secuencia seqob en índices del rango [1:numel(alf)]. Dichos índices los almacenarás en indseqob. Comprueba que indseqob contiene [2 3 1 3 3].

Paso 5

De un conjunto de entrenamiento hemos extraído las siguientes frecuencias de los símbolos. Éstas son, en el mismo orden que el alfabeto, [29 48 100]. Para codificar la secuencia que contiene los índices de los símbolos observados escribimos

```
counts=[29 48 100];
code=arithenco(indseqob,counts)
```

La salida es

code =

0 1 0 0 1 1 0 0 0 1 0 0 0 0 0 0

Lo que indica que hemos codificado la secuencia de 5 bytes indseqalf en sólo 16 bits

Paso 6

A continuación queremos decodificar la secuencia que contiene la codificación aritmética de nuestra secuencia original. Simplemente escribimos

```
indseqdec=arithdeco(code,counts,numel(indseqob));
seqdecf=alf(indseqdec);
fprintf('¿Coinciden original y comprimido 1(S) 0 (N)?,
%d\n',...
    isequal(seqob,seqdecf))
```

La salida es

¿Coinciden original y comprimido 1(S) 0 (N)?, 1

Paso 7

Vamos ahora a trabajar un poco los conceptos que hemos visto en teoría. Por simplicidad supondremos que las palabras de código son números enteros en el rango [1:256]. Ejecuta y entiende el siguiente trozo de código. Es importante que tengas claro cuál es el alfabeto

```
close all; clear all;
seqob=[1 1];
counts=[1 1];//el alfabeto 1 2
code=arithenco(seqob,counts);
fprintf('Longitud de la secuencia codificada %d\n',...
    numel(code))
```

La salida es

Longitud de la secuencia codificada 5

Paso 8

Compara en el paso 8 del fichero Practica03ApellidoNombre.pdf esta longitud con la que la teoría nos dice que es una cota superior al número de bits necesarios.

Escribe tus respuestas aquí.

1. como la secuencia es 1 1 y el dígito 1 está representado por 0.5 al haber dos dígitos 1 el intervalo resulta de 0 a 0.25 donde aplicando la fórmula nos resulta que $\log_2(1/0.25)$ es [2] +1= 3

Paso 9

Ejecuta ahora y entiende el siguiente trozo de código

```
close all; clear all;
seqob=[1 1];
counts=[50 50];
code=arithenco(seqob,counts);
fprintf('Longitud de la secuencia codificada %d\n',...
    numel(code))
```

La salida es

Longitud de la secuencia codificada 11

Paso 10

¿Por qué crees que ahora la longitud de la secuencia es 11 cuando las frecuencias [1 1] y [50 50] producen las mismas probabilidades?

Incluye la discusión en el paso 10 del fichero Practica03ApellidoNombre.pdf

Escribe tus respuestas aquí.

1. Porque en vez de haber 1 solo valor hay 50, y es dato también lo tiene que guardar.

Paso 11

Vamos ahora a analizar cómo funciona la codificación aritmética con distribuciones skew, es decir, distribuciones descompensadas. Comenzamos generando una secuencia de símbolos de un alfabeto con dos letras con probabilidades muy asimétricas. Esta secuencia nos servirá de conjunto de entrenamiento para calcular las frecuencias de cada símbolo. Observa que si no ha salido ningún 1 la secuencia de entrenamiento no nos servirá. A continuación construiremos su código aritmético

```
clear all;close all; clc;
maximo= 0.0;
minimo=0.0;
rng(0);
while maximo==minimo
```

```
seq=randsrc(1,100000,[1 2; 0.01 0.99]);
maximo=max(seq(:));
minimo=min(seq(:));
end
histo=histc(seq,[1 2]);//numero de veces que ha aparecido el
1,2
bar([1 2],histo);
```

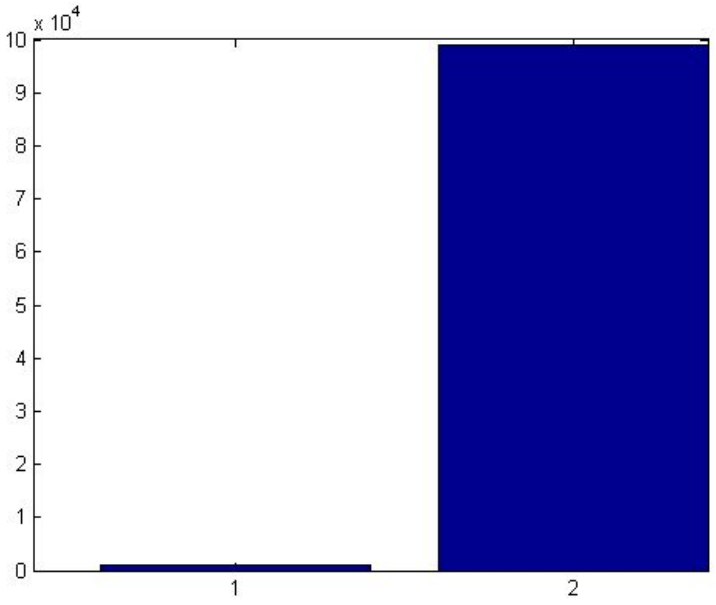
//010137 10

//1->0, 2->1, 3->3, 4->7, 5->10

//[po p1 p3 p7 p10]

//leer los simbolos meterlos en una matriz y codificar los indices, usar el find

El histograma que obtenemos es



Paso 12

Codificamos la secuencia observada, calculamos su longitud, la decodificamos y comprobamos que la secuencia original y la decodificada coinciden

```
code=arithenco(seq,histo);
fprintf('Longitud de la secuencia codificada
%d\n',numel(code));
indseqdec=arithdeco(code,histo,numel(seq));
fprintf('¿Coinciden original y comprimido 1(S) 0 (N)?,
%d\n',...
    isequal(seq,seqdec))
```

Las salidas son

```
Longitud de la secuencia codificada 8177
¿Coinciden original y comprimido 1(S) 0 (N)?, 1
```

Observa el tamaño en bits de la secuencia codificada y el tamaño original

Paso 13

Realiza un estudio de cómo evoluciona el número de bits por símbolo cuando generamos 10^i, i=1,2,3,4,5,6 símbolos siguiendo el proceso del paso 11 con probabilidades pr(1)=0.1 y pr(2)=0.9. Para cada uno de los 6 casos, realiza 5 simulaciones distintas y calcula el número medio de bits por símbolo como una media de las 5 simulaciones. Dibuja los 30 valores obtenidos así como las medias de los 5 valores para las 10, 100, 1000, 10000, 100000 y 1000000 simulaciones. Dibuja también una línea con el valor de la entropía de la fuente. ¿Obtienes algún resultado que en principio parezca incorrecto entre la entropía de la fuente y los bits por símbolo?.¿Cuál sería la explicación?

Incluye el código en el paso 13 del fichero Practica03ApellidoNombre.m y la discusión en el paso 13 de Practica03ApellidoNombre.pdf.

Escribe tus respuestas aquí.

1. La entropía parece que a mayores valores, va tomando un valor aproximado al 0.46, esto es debido a que la codificación aritmética no le influye el tamaño de la secuencia codificada, sino que este valor será más grande si el número de opciones es mayor.

Paso 14

Como habrás podido comprobar empíricamente, la codificación aritmética es más eficiente cuanto mayor sea la longitud de la secuencia a codificar, es decir, el número medio de bits necesarios para representar cada símbolo se acerca más a la entropía cuantos más caracteres codificamos. Comprobaremos de nuevo empíricamente este hecho usando los ficheros de texto contenidos en el fichero `textobinario.zip`.

Los ficheros de texto en `textobinario.zip` contienen sólo dos caracteres, el carácter '0' y el carácter '1'. El nombre de cada fichero es `texto`*X*`.txt` donde *X* representa el número de caracteres en el fichero. Observa que los caracteres ‘0’ y ‘1’ vas a tener que codificarlos como 1 y 2.

Incluye en el paso 14 de Practica03ApellidoNombre.m el código para

1. codificar y decodificar cada uno de estos ficheros,
2. comprobar que la secuencia original y decodificada coinciden,
3. dibujar los histogramas de los símbolos en cada fichero.

Incluye los histograma, la tabla que contiene el número de bits por símbolo para cada fichero y las conclusiones que puedas extraer en el paso 14 de Practica03ApellidoNombre.pdf.

Escribe tus respuestas aquí.

Nº Caracteres	Bits/símbolo
10	1.6
100	1.07
1000	1.011
10000	1.0015
100000	1.00018
1000000	1.000021

Paso 15

Utiliza codificación aritmética para codificar cada uno de ficheros de texto constitucion española.txt, Fundacion e Imperio - Isaac Asimov.txt y Cinco semanas en globo - Julio Verne.txt (dentro de Prácticas - Datos - texto.zip).

Incluye en el paso 15 de Practica03ApellidoNombre.m el código para

1. codificar y decodificar cada uno de estos ficheros,
2. comprobar que la secuencia original y decodificada coinciden,
3. dibujar los histogramas de los símbolos en cada fichero,
4. calcular el factor de compresión obtenida para cada uno de los ficheros,
5. calcular el número de bits por símbolo para cada fichero.

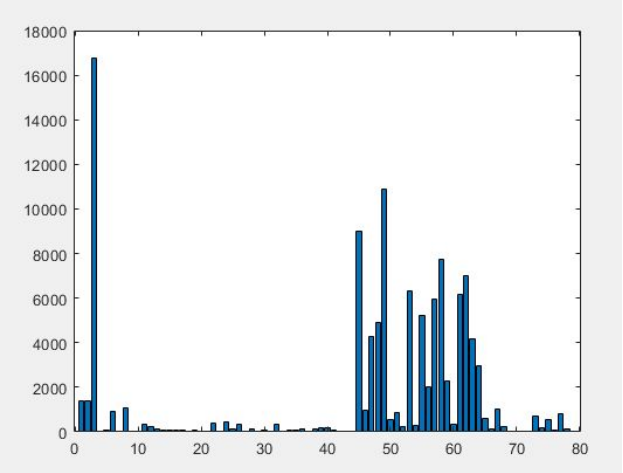
En el paso 15 de Practica03ApellidoNombre.pdf incluye

1. los histogramas,
2. completa las tablas adjuntas, para la codificación Huffman no incluyas el tamaño de la cabecera y
3. realiza una comparación crítica de los resultados obtenidos usando codificación Huffman y aritmética

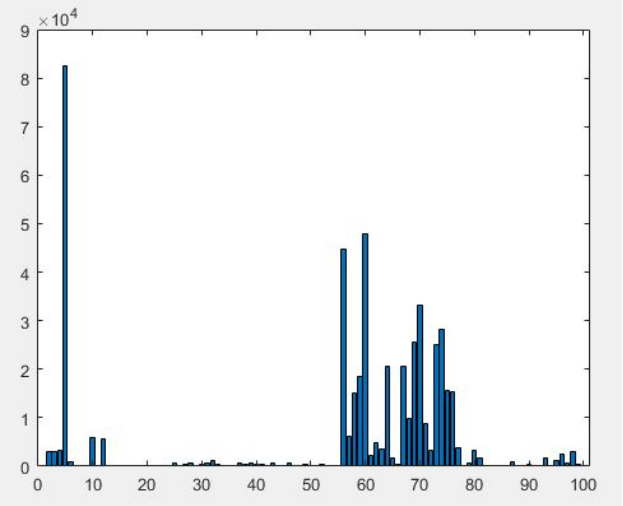
Escribe tus respuestas aquí.

1.

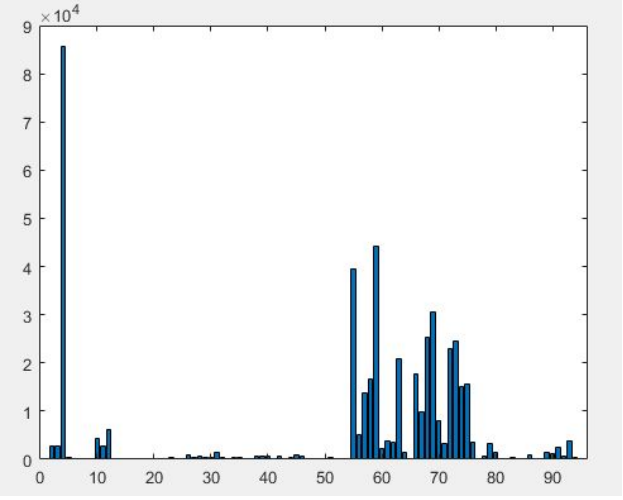
A. Constitución española



B. Fundación e Imperio



C. Cinco semanas en globo



Fichero\Huffman	Tamaño fichero original	Factor de compresión	Bit/símbolo
Constitución española	112246	1.771140	4.516865
Fundación e Imperio	461298	1.782608	4.4878068
Cinco semanas en globo	487020	1.772755	4.5127510

Fichero \aritmética	Tamaño fichero original	Factor de compresión	Bit/símbolo
Constitución española	112246	1.7824460	4.488213
Fundación e Imperio	461298	1.800616	4.442924
Cinco semanas en globo	487020	1.7878000	4.4747737

3. Los resultados obtenidos muestran como la codificación por aritmética es mejor que la codificación Huffman aunque esta diferencia es del factor de decimales, en ficheros de más tamaño esto puede significar una gran ventaja para la codificación aritmética

Paso 16

Utiliza codificación aritmética sobre los ficheros de imágenes ptt1.pbm, ptt4.pbm, ptt8.pbm (dentro de material complementario - Datos para prácticas - imgs_binarias.zip) y camera.pgm, bird.pgm y bridge.pgm (dentro de Prácticas - Datos - imgs_grises.zip).

Incluye en el paso 16 de Practica03ApellidoNombre.m el código para

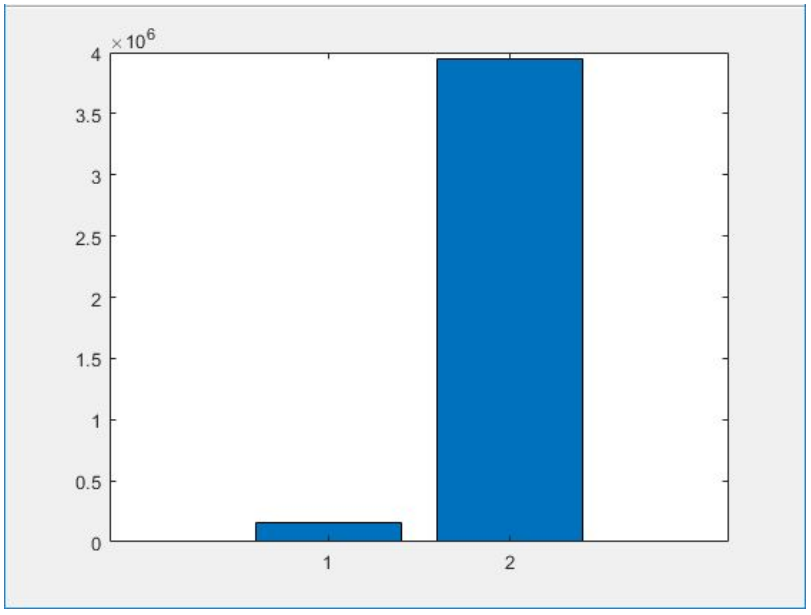
1. codificar y decodificar cada uno de estos ficheros,
2. comprobar que la secuencia original y decodificada coinciden,
3. dibujar los histogramas de los símbolos en cada fichero,
4. calcular el factor de compresión obtenida para cada uno de los ficheros,
5. calcular el número de bits por símbolo para cada fichero.

En el paso 16 de Practica03ApellidoNombre.pdf incluye

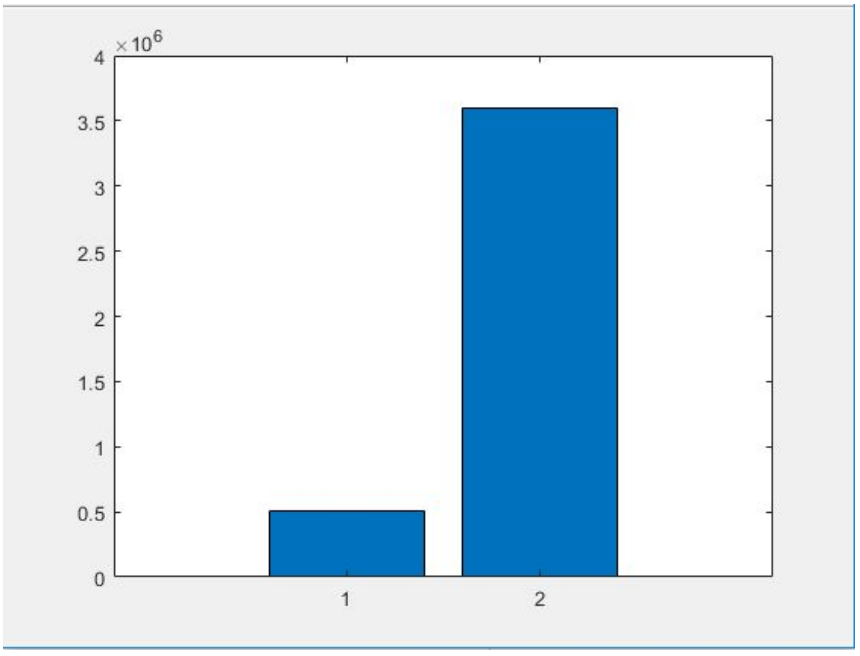
1. los histogramas,
2. completa las tablas adjuntas, para la codificación Huffman no incluyas el tamaño de la cabecera y
3. realiza una comparación crítica de los resultados obtenidos usando codificación Huffman y aritmética

Escribe tus respuestas aquí.

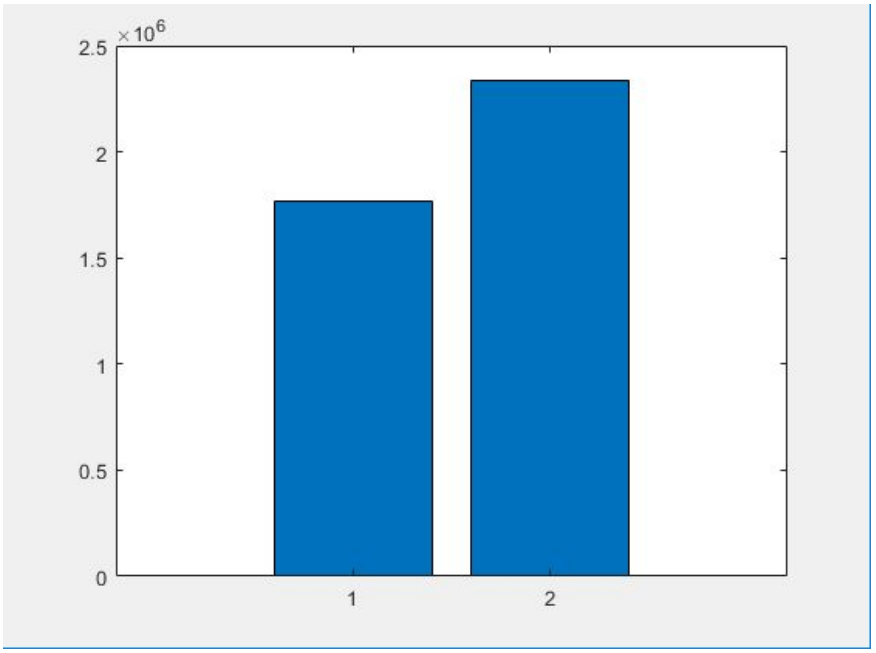
1. Histogramas
    a. ptt1



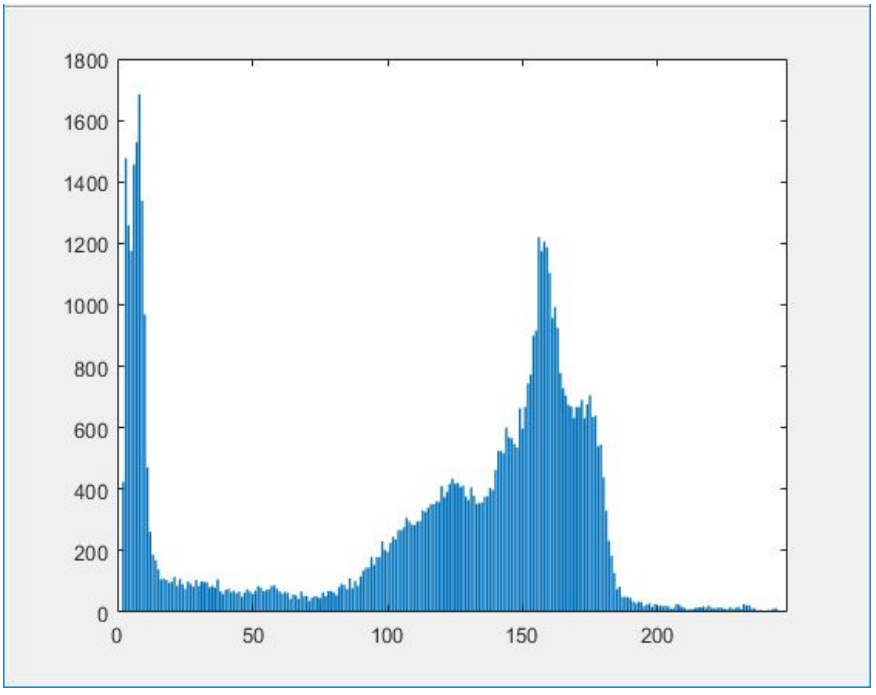
    b. ptt4



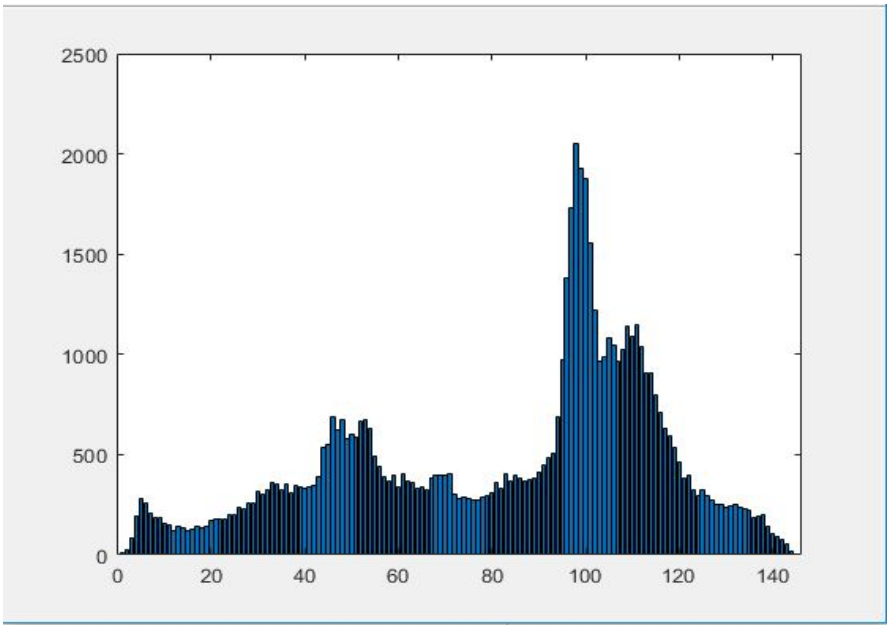
    c. ptt8



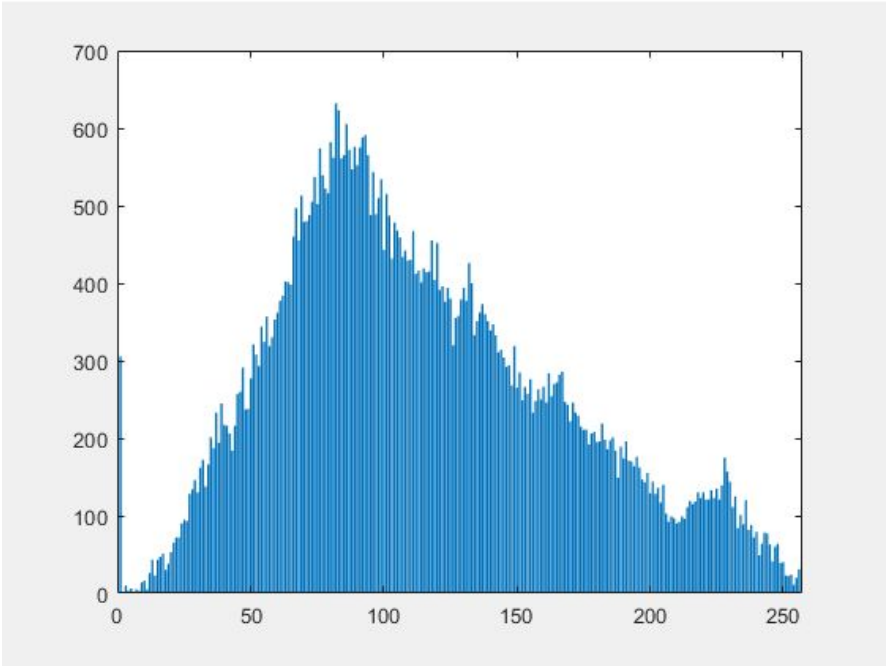
d. camera



e. bird



f. bridge



2. Tabla

Fichero \Huffman	Tamaño fichero original	Factor de compresión	Bits/símbolo
ptt1.pbm	4105728	8.00000	1
ptt4.pbm	4105728	8.00000	1
ptt8.pbm	4105728	8.00000	1
camera.pgm	65536	1.135578	7.0449225
bird.pgm	65536	1.17619	6.801636
bridge.pgm	65536	1.03982	7.693604

Fichero \Aritmética	Tamaño fichero original	Factor de compresión	Bits/símbolo
ptt1.pbm	4105728	34.3984	0.2325687
ptt4.pbm	4105728	14.784228	0.5411165
ptt8.pbm	4105728	8.1142461	0.9859202
camera.pgm	65536	1.141225	7.010010
bird.pgm	65536	1.1808719	6.774658
bridge.pgm	65536	1.043185	7.668823

3. Los tres primero archivos denotan que el número de datos es poco, siendo la codificación aritmética muy buena, consiguiendo un factor de compresión que el método de Huffman no podría alcanzar. Aunque en los tres siguientes al ser un mayor número de datos diferentes ambos métodos obtienen resultados muy similares aunque la codificación aritmética permanece siendo mejor por decimas.

Paso 17

No hemos discutido en clase qué cabecera debemos incluir en el fichero codificado para que el decodificador sea capaz de reconstruir el fichero original. Suponiendo que, como mucho, las letras del alfabeto son 256, ¿qué cabecera incluirías?. No olvides incluir las frecuencias o probabilidades si lo consideras necesario. Podemos incluir la longitud de la secuencia a decodificar, ¿habría alguna forma de no tener que incluir el número de símbolos a decodificar?.

Incluye la discusión en el paso 17 de Practica03ApellidoNombre.pdf.

Escribe tus respuestas aquí.

1. Sería necesario saber las frecuencias de cada símbolo, al igual que sería necesario conocer el alfabeto sobre el que estamos trabajando. Por tanto en la cabecera solo habría que incluir las frecuencias y el símbolo al que pertenece.
2. Si, teniendo un símbolo solo para expresar que se ha terminado la decodificación.