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 contendrá:

- 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 porqué 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 indseqob 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];
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.

Longitud media de bits por símbolo → l = log2(1/(alto-bajo))+1

La probabilidad de cada símbolo en el ejemplo es de 0.5, por lo que el intervalo que obtenemos es el siguiente → [0.25, 0.5]. Este intervalo cumple que 0.5 * 0.5 (∏probabilidades) = 0.5 – 0.25 (alto - bajo) = 0.25.

Aplicando esto tenemos que l = log2(1/(alto-bajo))+1 = log2(4)+1 = 3 = H(secuencia) + 2.

La longitud obtenida es 5 y no 3. Esto es debido a que durante la codificación, el método 'arithenco' ha modificado la longitud. En la siguiente imagen se muestra el fragmento del código de 'arithenco' que realiza el cálculo.

```
% Compute the Word Length required.
total_count = cum_counts(end);
N = ceil(log2(total_count)) + 2;

% Initialize the lower and upper bounds.
dec_low = 0;
dec_up = 2^N-1;
E3_count = 0;

% Obtain an over estimate for the length of CODE and initialize CODE
code_len = length(seq) * ( ceil(log2(length(counts))) + 2 ) + N;
code = zeros(1, code_len);
code_index = 1;
```

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.

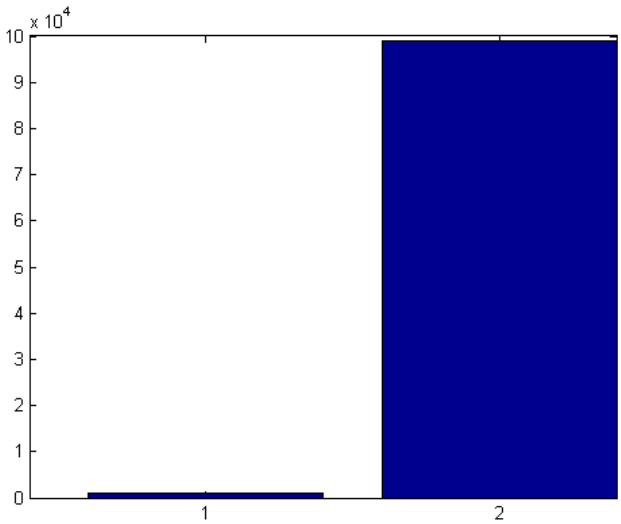
La longitud debería ser la longitud teórica del paso 8, es decir, 3. Debido a la codificación que realiza 'arithrnco' este valor difiere del teórico (3) que, en este caso, es 11 ya que trata con más "tiradas" (50) que en el paso 8.

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]);
bar([1 2],histo);
```

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

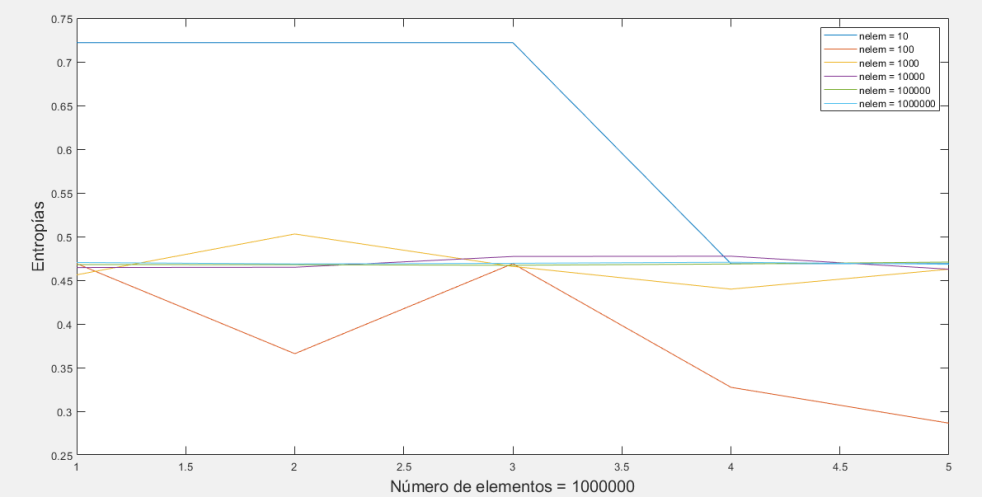
Paso 13

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 de alguna simulación?, ¿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.

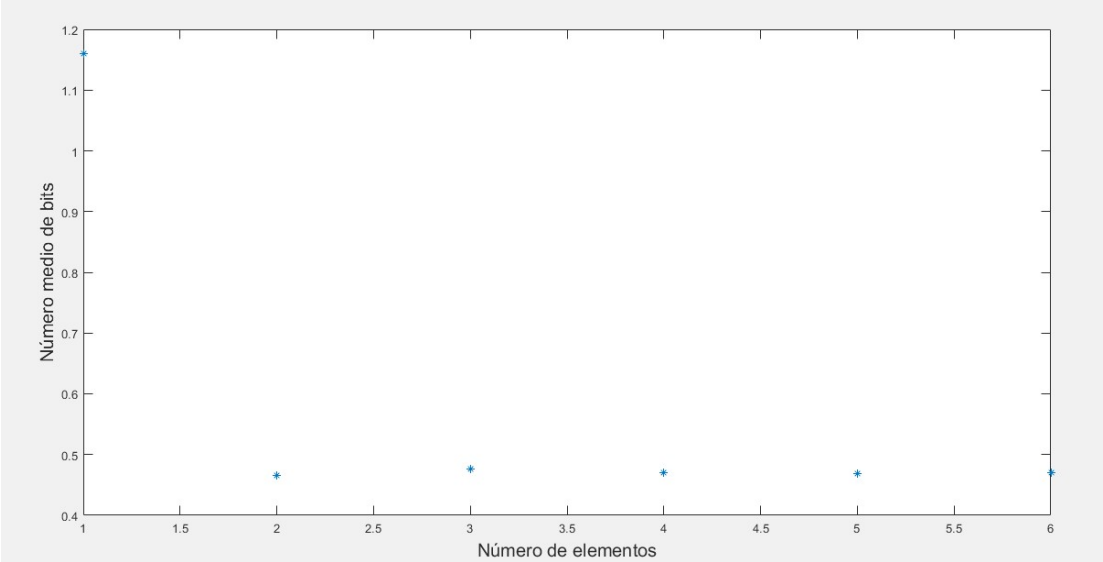
Escribe tus respuestas aquí.

En la siguiente imagen se muestran las líneas de las entropías para cada uno de los 6 casos (5 valores por cada uno de ellos).



La siguiente curva de valores muestra el número medio de bits que se ha necesitado para cada experimento. Cada uno de los seis casos cuenta con cinco ejecuciones diferentes (se ha requerido retirar la función rng(0)) por lo que cada punto es la media de dichos valores.

El resultado obtenido muestra que el valor medio se acerca a la entropía conforme aumenta el número de valores de la secuencia original.



Por ejemplo, para un experimento (diferente al de las imágenes anteriores), se han obtenido los siguientes resultados:

$N=10^1$

Entropías: [0.469, 0.469, 0.469, 0.469, 0.72193] / Media de bits: 1.060000e+00

$N=10^2$

Entropías: [0.49992, 0.36592, 0.68008, 0.32744, 0.40218] / Media de bits: 5.360000e-01

$N=10^3$

Entropías: [0.45619, 0.51186, 0.52068, 0.41956, 0.48155] / Media de bits: 4.888000e-01

$N=10^4$

Entropías: [0.46645, 0.46294, 0.46326, 0.46486, 0.47089] / Media de bits: 4.672200e-01

$N=10^5$

Entropías: [0.46874, 0.46696, 0.4676, 0.4668, 0.46969] / Media de bits: 4.681420e-01

$N=10^6$

Entropías: [0.46889, 0.46948, 0.47042, 0.46702, 0.46883] / Media de bits: 4.689504e-01

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`.

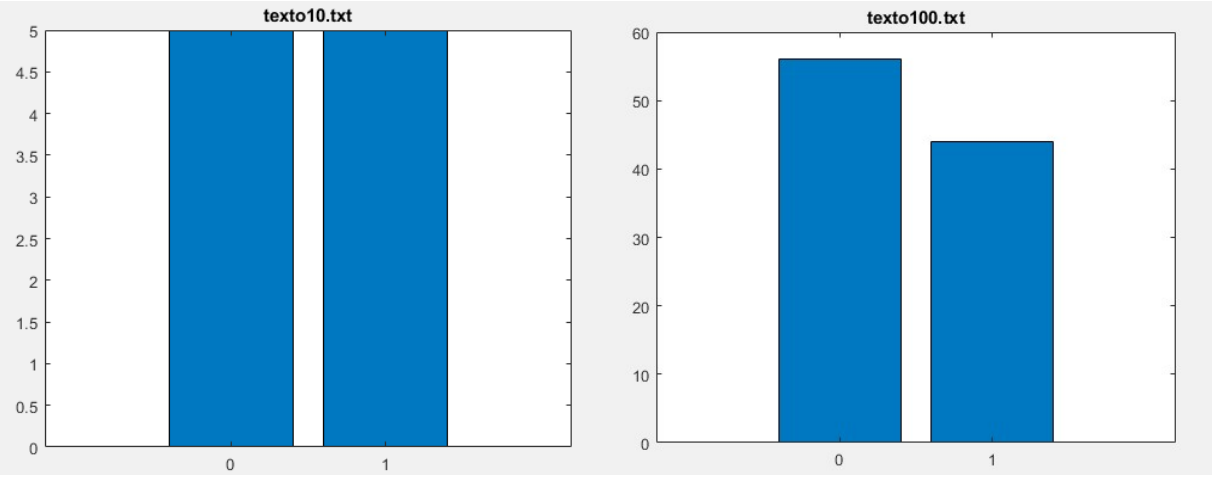
Los ficheros de texto en `textobinario.zip` contienen sólo dos caracteres, el carácter '0' y el carácter '1'. El nombre de cada fichero es `texto`*X*`.txt` donde *X* representa el número de caracteres en el fichero. Observa que los caracteres '0' y '1' vas a tener que codificarlos como 1 y 2.

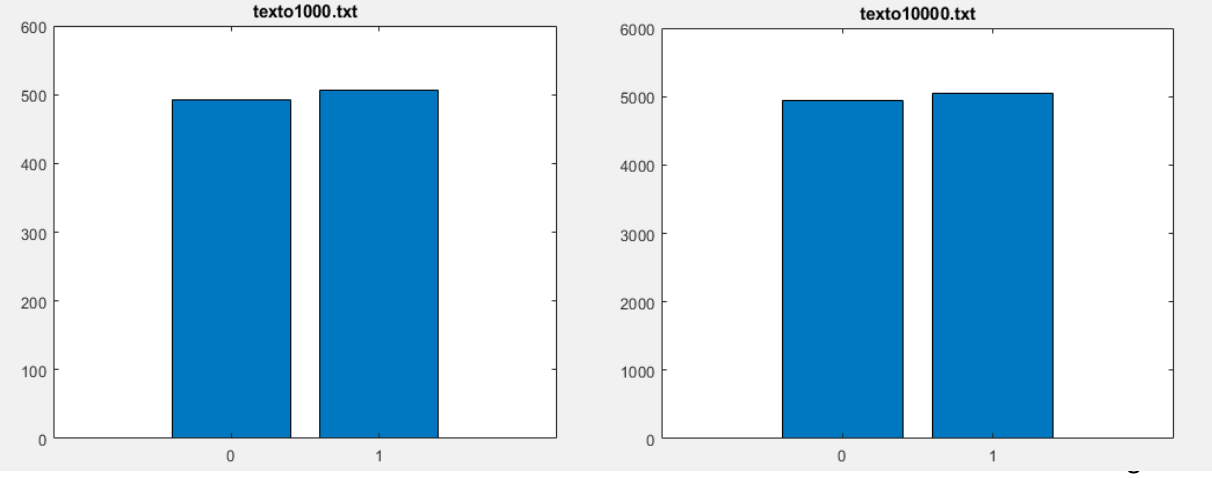
Incluye en el paso 14 de Practica03ApellidoNombre.m el código para

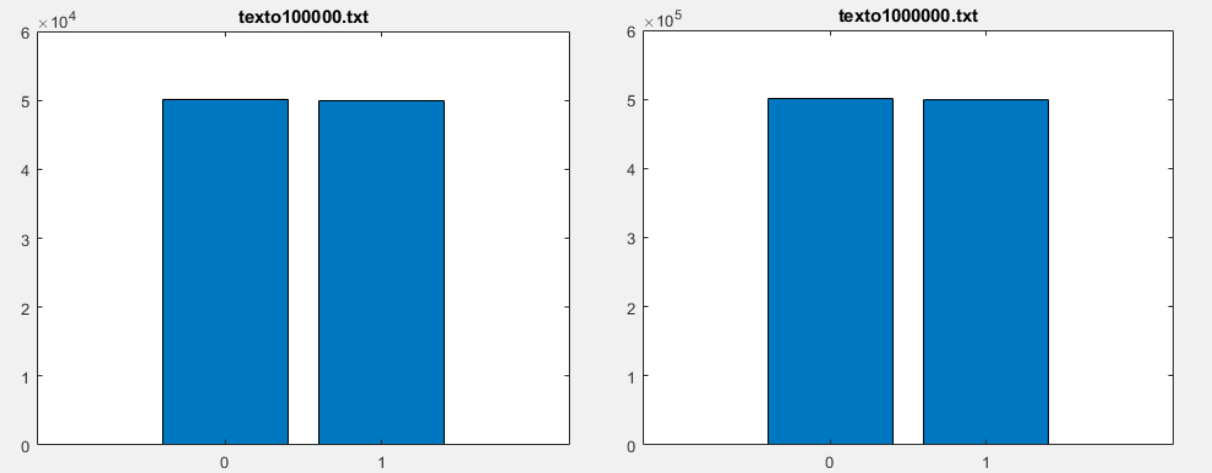
1. codificar y decodificar cada uno de estos ficheros,
2. comprobar que la secuencia original y decodificada coinciden,
3. dibujar los histogramas de los símbolos en cada fichero.

Incluye los histogramas, 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	1.0014
100000	1.00018
1000000	1.00002

Se puede observar en el resultado obtenido lo comentado en los pasos anteriores. A mayor longitud de secuencia, mejor resultado ofrece la codificación aritmética. El número de bits necesario se acerca cada vez más a la entropía, que en este caso es 1 ya que hay dos símbolos con una probabilidad de aparición del 50% casa uno.

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).

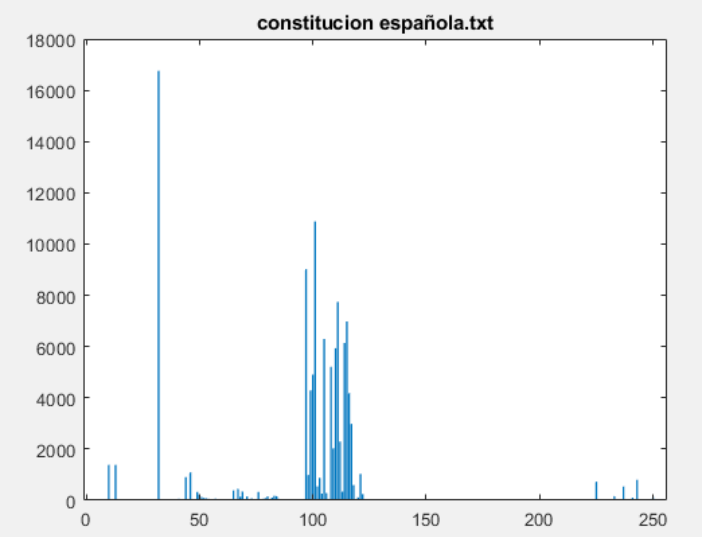
Incluye en el paso 15 de Practica03ApellidoNombre.m el código para

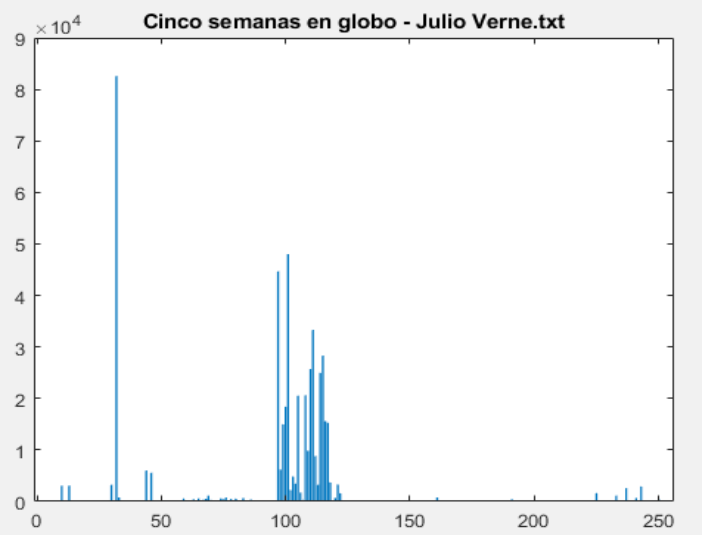
1. codificar y decodificar cada uno de estos ficheros,
2. comprobar que la secuencia original y decodificada coinciden,
3. dibujar los histogramas de los símbolos en cada fichero,
4. calcular el factor de compresión obtenida para cada uno de los ficheros,
5. calcular el número de bits por símbolo para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

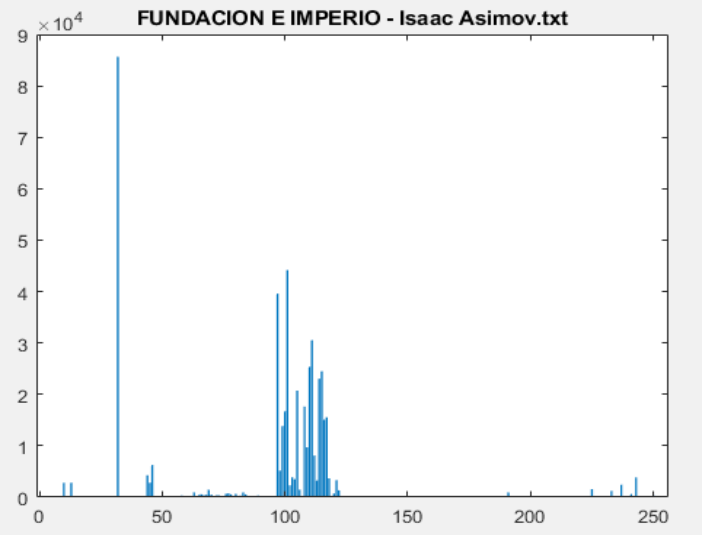
En el paso 15 de Practica03ApellidoNombre.pdf incluye

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.







Fichero\Huffman	Tamaño fichero original	Factor de compresión	Bit/símbolo
Constitución española	112246 bytes	1.751244	4.516856
Fundación e Imperio	461298 bytes	1.776676	4.487791
Cinco semanas en globo	487020 bytes	1.766908	4.512739

Fichero \aritmética	Tamaño fichero original	Factor de compresión	Bit/símbolo
Constitución española	112246 bytes	1.782464	4.488169
Fundación e Imperio	461298 bytes	1.800619	4.442915
Cinco semanas en globo	487020 bytes	1.787800	4.474773

El número de bits por símbolo requeridos es menor en la codificación aritmética. Como hemos visto en pasos anteriores, este tipo de codificación es más eficiente para cantidades más grandes de elementos mientras que, en Huffman, esto no sucede por lo que el resultado es el deseado.

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).

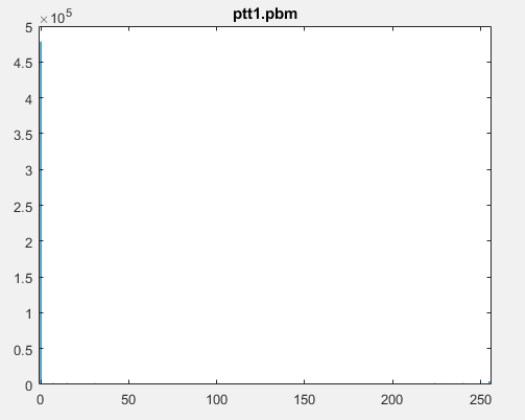
Incluye en el paso 16 de Practica03ApellidoNombre.m el código para

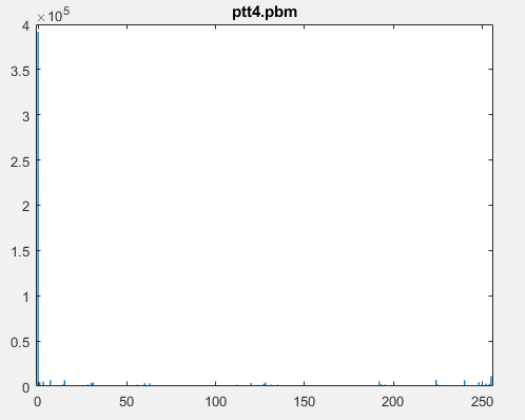
1. codificar y decodificar cada uno de estos ficheros,
2. comprobar que la secuencia original y decodificada coinciden,
3. dibujar los histogramas de los símbolos en cada fichero,
4. calcular el factor de compresión obtenida para cada uno de los ficheros,
5. calcular el número de bits por símbolo para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

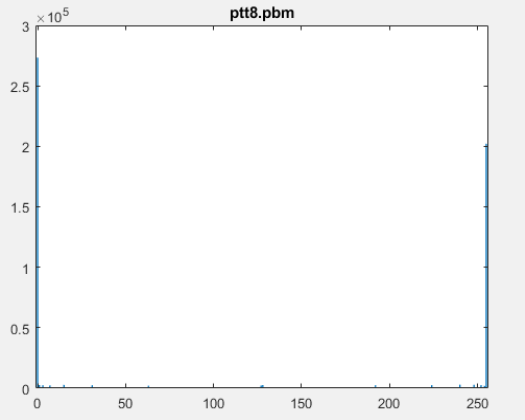
En el paso 16 de Practica03ApellidoNombre.pdf incluye

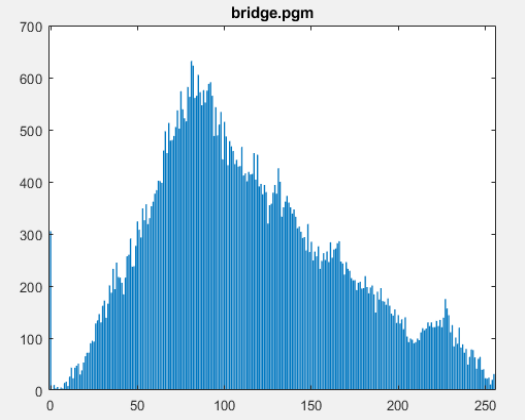
1. los histogramas,
2. completa las tablas adjuntas, para la codificación Huffman no incluyas el tamaño de la cabecera y
3. realiza una comparación crítica de los resultados obtenidos usando codificación Huffman y aritmética

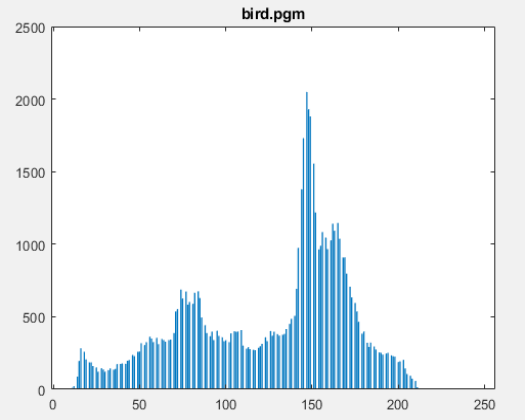
Escribe tus respuestas aquí.

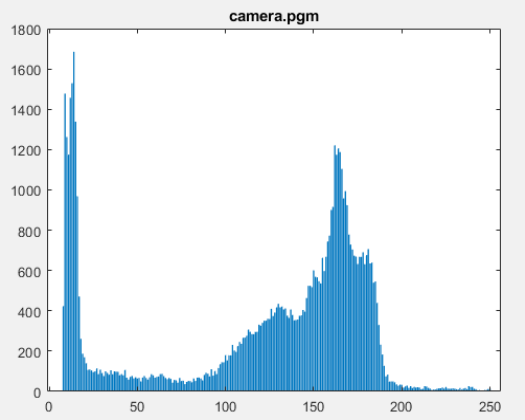












Fichero \Huffman	Tamaño fichero original	Factor de compresión	Bits/símbolo
ptt1.pbm	513249 bytes	5.870063	1.342080
ptt4.pbm	513249 bytes	3.526103	2.245082
ptt8.pbm	513249 bytes	4.440946	1.784572
camera.pgm	65551 bytes	1.093227	7.045293
bird.pgm	65551 bytes	1.148627	6.802276
bridge.pgm	65551 bytes	1.002984	7.693902

Fichero \Aritmética	Tamaño fichero original	Factor de compresión	Bits/símbolo
ptt1.pbm	513249 bytes	1.146285	6.979069
ptt4.pbm	513249 bytes	3.946677	2.027022
ptt8.pbm	513249 bytes	4.980353	1.606312
camera.pgm	65551 bytes	1.141151	7.010465
bird.pgm	65551 bytes	1.180735	6.775442
bridge.pgm	65551 bytes	1.043148	7.669097

Los resultados reflejan obtenidos en este paso reflejan lo comentado en el paso anterior.

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.

La cabecera contendría el tipo de dato de los símbolos, la longitud de la secuencia codificada, el alfabeto de los símbolos (elementos utilizados de la secuencia original), la probabilidad de cada elemento de la secuencia de aparecer en el alfabeto y el número de símbolos a decodificar.