TRANSMISIÓN DE DATOS 2007/08			
Práctica 1: Codificación Huffman		**Grupo**	
		Puesto	
Apellidos, nombre		**Fecha**	
Apellidos, nombre		**22/24/25 de Octubre de 2007**	

El objetivo de esta práctica es el de familiarizar al alumno con la codificación de fuente (sin pérdidas) mediante el algoritmo *Huffman*.

En el fichero `huffman.zip` se encuentra el código y ficheros de prueba necesarios para la realización de la práctica. Para ello descomprima y copie los ficheros al directorio de realización de la práctica.

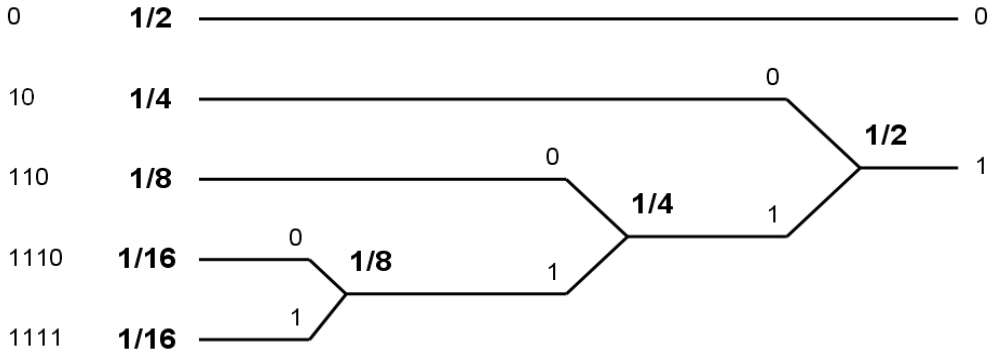
Justo antes de finalizar la práctica, comprima el proyecto en un fichero TxDatosP1GXzz.zip (siendo X el grupo –A, B ó C-, y zz el número de pareja) conéctese al sistema de entrega de prácticas de la Intranet y entréguelo en el grupo que corresponda (A, B o C). Guárdese adicionalmente una **copia personal**, para posibles futuras reutilización del código en prácticas posteriores. Recuerde borrar su trabajo del ordenador del puesto de prácticas.

1 Introducción

La codificación *Huffman* es uno de los métodos clásicos de codificación de fuente de la familia de los métodos estadísticos, esto es, aquellos que necesitan conocer la distribución probabilística de la fuente.

El algoritmo básico consiste en, tras ordenar los símbolos de mayor a menor en probabilidad, ir juntando parejas de menor probabilidad formando un árbol. Cuando solamente haya dos raíces, se asigna los símbolos 0 y 1 (o 1 y 0) a cada raíz, y se itera hacia atrás.

Dada, por ejemplo, una fuente F con 5 símbolos de probabilidades {1/2, 1/4, 1/8, 1/16, 1/16} se podría construir el siguiente diagrama de árbol.



En cada etapa los dos elementos con menor probabilidad se unen y se obtiene un elemento resultante cuya probabilidad es la suma de las dos anteriores. Cada vez que se realiza una unión de dos elementos se asigna a cada uno de ellos un ‘1’ o un ‘0’ (el resultado dependerá de que criterio de asignación se aplique. El proceso termina cuando únicamente quedan dos elementos y también a cada uno de ellos se le asigna un ‘1’ o un ‘0’. Finalmente para conocer el código asociado a cada probabilidad se recorre el árbol en sentido inverso y se concatenan los unos o ceros por los que se va pasando hasta llegar al principio de cada ramificación.

2 Códigos Huffman

En este apartado se implementarán funciones para la obtención de los códigos de símbolo así como de los datos estadísticos (entropía y longitud media) para una colección de símbolos. En el fichero de la práctica se proporciona la función Matlab `huffmanRecursivo` con la siguiente especificación:

```
function [codigos] = huffmanRecursivo(P)
```

- **Recibe:**
  - P == Un vector de probabilidades.
- **Devuelve:**
  - Codigos == Un tipo 'cell' (ver ayuda sobre tipos 'cell' al final del guión) en el que cada posición contiene un vector fila con el código *Huffman* asignada a la probabilidad correspondiente (según el orden del vector de probabilidades).

Haciendo uso de esta función **escriba una función** Matlab `asignarCodigosHuffman` con la siguiente especificación:

```
function [tablaCodigos]=asignarCodigosHuffman(listaSimbolos, listaProbabilidades)
```

- **Recibe:**
  - listaSimbolos == Un vector con una lista de símbolos (usaremos números representando cada símbolo)
  - listaProbabilidades == Un vector con la lista de probabilidades de cada símbolo de la lista anterior.
- **Devuelve:**
  - tablaCodigos == Un tipo 'cell' con 3 columnas: En la primera columna incluirá cada uno de los símbolos recibidos en 'listaSimbolos', en la segunda columna los códigos Huffman correspondientes obtenidos mediante la función `huffmanRecursivo` y en la tercera las probabilidades originales recibidos en 'listaProbabilidades'.

Tomando como referencia la función `imprimirTablaCodigos` suministrada,

```
function [] = imprimirTablaCodigos(tablaCodigos)
```

- **Recibe:**
  - tablaCodigos == Un cell resultado de la llamada a la función `asignarCodigosHuffman` implementada anteriormente (ver especificación).

que imprime la información relativa a la tabla de códigos recibida (símbolos, probabilidades y códigos asignados), **escriba una función Matlab** `estadisticasCodigo` con la siguiente especificación:

```
function [entropía, longitudMedia] = estadisticasCodigo(tablaCodigos)
```

- **Recibe:**
  - tablaCodigos == Un cell resultado de la llamada a la función `asignarCodigosHuffman` implementada anteriormente (ver especificación).
- **Devuelve:**
  - entropia == La entropía de la fuente correspondiente a la tabla de códigos recibida
  - longitudMedia == La longitud media del código especificado en la tabla de códigos recibida.

2.1 Ejercicio 1: Fuente A

Escriba un **script Matlab con nombre 'Ejercicio1.m'** en el que utilizando las funciones desarrolladas en el apartado anterior se calculen los códigos *Huffman* para la fuente A con 4 símbolos `[1 2 3 4]` y probabilidades asociadas `[0.6 0.2 0.1 0.1]`, se muestren por pantalla los símbolos y códigos asociados así como la entropía y longitud media asociadas.

Calcule analíticamente la entropía, el código *Huffman* y la longitud media del código para la fuente A.

El código {0, 10,110,111} también es un código *Huffman* válido para la fuente A. ¿Cual es el motivo de que se puedan generar diferentes códigos *Huffman* válidos para la misma fuente?

2.2 Ejercicio 2: Fuente B

Dada la fuente B con 4 símbolos `[1 2 3 4]` y probabilidades asociadas `[0.5 0.3 0.15 0.05]` repita los pasos del ejercicio 1 escribiéndolos en un **script Matlab con nombre 'Ejercicio2.m'**

Calcule analíticamente la entropía, el código *Huffman* y la longitud media del código para la fuente B.

2.3 Ejercicio 3: Fuente C

Dada la fuente C con 8 símbolos `[1 2 3 4 5 6 7 8]` y probabilidades asociadas `[0.3 0.15 0.25 0.2 0.05 0.025 0.015 0.01]` repita los pasos de los ejercicios anteriores escribiéndolos en un **script Matlab con nombre 'Ejercicio3.m'**

Calcule analíticamente la entropía, el código Huffman y la longitud media del código para la fuente C.

A la vista de los resultados para las fuentes A, B y C (analíticos y del programa) comente el algoritmo implementado por la función `matlab` que crea el código *Huffman* y las diferencias o semejanzas encontradas al realizar los cálculos analíticamente.

3 Codificador Huffman

En este apartado se llevará a cabo la implementación de un codificador *Huffman* sencillo. Los pasos necesarios para llevar a cabo la codificación serán los siguientes:

- Lectura de un fichero de disco y generación de estadísticas de símbolos.
- Asignación de códigos *Huffman*.
- Sustitución de símbolos originales por sus códigos *Huffman* asociados.

3.1 Ejercicio 4: Obtención de códigos Huffman

Para la realización de este ejercicio se considerarán como posibles símbolos todos los valores numéricos entre 0 y 255. De esta forma se podrá leer cualquier fichero de disco byte a byte (1byte = valor entre 0 y 255). En el contenido de los ficheros para la práctica se incluye la siguiente función:

```
function [lectura] = lecturaFichero(nombreFichero)
```

- **Recibe:**
  - nombreFichero== Nombre de fichero a leer de disco. Se debe especificar entre comillas simples (p.ej. 'miFichero.txt').
- **Devuelve:**
  - lectura == Un vector con los valores enteros correspondientes a todos los bytes del fichero leido. (Se pueden visualizar como caracteres haciendo un casting a char, ver explicación de códigos ASCII al final del guión)

Escriba una función Matlab para el cálculo de probabilidades de símbolos a partir de la siguiente especificación:

```
function [probabilidades] = probabilidadesSimbolos(lectura)
```

- **Recibe:**
  - lectura == resultado de la lectura de un fichero con la función `lecturaFichero`.
- **Devuelve:**
  - probabilidades == Un vector con las probabilidades de aparición de los 256 posibles símbolos (por tanto el tamaño del vector será 256) calculadas a partir del vector recibido 'lectura'.

Escriba un **script Matlab con nombre 'Ejercicio4.m'** en el se lleve a cabo la lectura del fichero suministrado 'PruebaEstadísticas.txt' y se calcule la probabilidad de cada posible símbolo (valores entre 0 y 255). Debe tenerse en cuenta que mientras las posibles lecturas serán valores comprendidos entre 0 y 255 la tabla donde se almacenen se indexará con valores entre 1 y 256. A continuación haga uso de las funciones `huffmanRecursivo`, `asignarCodigosHuffman`, `imprimirTablaCodigos` y `estadisticasCodigo` para asignar y mostrar los códigos *Huffman* correspondientes a cada símbolo. Indique los códigos *Huffman* asignados a las letras 'a', 'b', 'c', 'd' (valores enteros correspondientes: 97,98,99,100):

3.2 Ejercicio 5: Codificación

En este apartado se implementará el proceso por el que los bytes del fichero original se codificarán sustituyéndolos por sus códigos *Huffman* correspondientes.
En el fichero de la práctica se incluye la función `devolverCodigoHuffman` con la siguiente especificación:

```
function [codigoHuffman] = devolverCodigoHuffman(tablaCodigos, simbolo)
```

- **Recibe:**
  - tablaCodigos == Un cell resultado de la llamada a la función `asignarCodigosHuffman` implementada anteriormente (ver especificación). Incluye una lista de símbolos, sus probabilidades y códigos *Huffman* asociados.
  - símbolo == valor numérico de un símbolo del que se quiere conocer su código *Huffman* asociado
- **Devuelve:**
  - codigoHuffman == Vector fila en el que se especifica, en una posición por cada bit, el código *Huffman* asociado al símbolo recibido en función de la tabla de códigos recibida.

Haciendo uso de esta función **escribir una funcion** `codificaHuffman` con la siguiente especificación:

```
function [bitsHuffman] = codificaHuffman(simbolos, tablaCodigos)
```

- **Recibe:**
  - simbolos == Un vector fila especificando un conjunto de símbolos que se quieren codificar. En nuestro caso los posibles símbolos serán los valores entre 0 y 255.
  - tablaCodigos == Un cell resultado de la llamada a la función `asignarCodigosHuffman` implementada anteriormente (ver especificación). Incluye una lista de símbolos, sus probabilidades y códigos *Huffman* asociados.
- **Devuelve:**
  - bitsHuffman == Vector fila en el que se especifica, en una posición por cada bit, los códigos *Huffman* asociados a todos los símbolos recibidos (en el mismo orden).

A partir de los resultados y funciones anteriores escriba un **script Matlab con nombre 'Ejercicio5.m'** en el que:

- Se lea de disco el fichero 'MensajeCorto.txt' (incluido en el fichero de la práctica).
- Se extraigan sus estadísticas y se calculen los códigos *Huffman* asociados.
- Se codifique dicho fichero con las funciones implementadas en este apartado.

Comente los resultados de la codificación. ¿Cuántos bits son necesarios para codificar el fichero? ¿Cuántos bytes se necesitarían para codificarlo? Compare la longitud del fichero original y los bytes necesarios teóricamente para codificarlo. ¿A qué se deben las diferencias?

3.3 Ejercicio 6: Pruebas de codificación I

Realice un **script Matlab con nombre 'Ejercicio6.m'** en el que se lleve a cabo todo el proceso de lectura y codificación de la imagen 'ImagenPlana.bmp'. En adelante, para la lectura de imágenes habrá de utilizar la función `lecturaFichero.` Compare el número de bits/bytes del fichero original con respecto a los necesarios para la codificación *Huffman*. Explique los resultados.

3.4 Ejercicio 7: Pruebas de codificación II

Realice un **script Matlab con nombre 'Ejercicio7.m'** en el que se lleve a cabo todo el proceso de lectura y codificación de la imagen 'ImagenVariación.bmp'. Compare el número de bits/bytes del fichero original con respecto a los necesarios para la codificación *Huffman*. Compare los resultados con los del ejercicio 6. ¿A que se deben las diferencias? Explique los resultados.

3.5 Ejercicio 8: Pruebas de codificación III

Realice un **script Matlab con nombre 'Ejercicio8.m'** en el que lea el fichero 'ImagenVariación.bmp' se extraigan sus estadísticas y la tabla de códigos *Huffman*. Realice una segunda lectura del fichero 'ImagenPlana.bmp'. Codifique la lectura de esta segunda imagen con la tabla de códigos *Huffman* extraida para el primer fichero. Compare el tamaño del fichero original con el resultado de la codificación ¿Cómo explicaría los resultados?

4 Ayudas

4.1 Tipos Cell en MATLAB

Los tipos cell en Matlab representan una matriz en la que cada posición puede, a su vez, contener matrices de tamaño variable.

Para la creación de una variable tipo cell se utilizará el comando *cell*:

- CELL(N) is an N-by-N cell array of empty matrices.
- CELL(M,N) or CELL([M,N]) is an M-by-N cell array of empty matrices.

(extraido de la ayuda de Matlab)

Importante: Para acceder a una posición concreta de un tipo cell se indexará utilizando llaves '{' '}' en vez de paréntesis. Es decir, si tenemos una variable tipo cell de 3x3 llamada 'celda' para acceder a sus posiciones utilizaremos 'celda{1,1}', 'celda{3,2}' o cualquier otra posición valida.

Podríamos por ejemplo almacenar una matriz completa en una posición de un cell:

```
variableCELL = cell(3,2);
variableCELL{1,1} = [1 2;3 4]
```

4.2 Codigos ASCII

(extraido de la wikipedia: http://es.wikipedia.org/wiki/C%C3%B3digo_de_caracteres_de_8_bits y http://es.wikipedia.org/wiki/ASCII)

Con un **código de caracteres de 8 bits** (1 byte) se pueden representar hasta $2^8 = 256$ caracteres diferentes.Existe un código que desde que fue definido en 1963, ha sido adoptado como el estándar para la transmisión de datos. Este código denominado ASCII (*American Standard Code for Information Interchange*) permite representar hasta 128 caracteres diferentes, para ello necesita 7 bits ($2^7 = 128$ combinaciones). Normalmente el código ASCII se extiende a 8 bits (1 byte) añadiendo un bit de control, llamado bit de paridad.

Tabla de Códigos ASCII:

0		32		64	@	96	`	128	Ç	160	á	192	└	224	α
1	☺	33	!	65	A	97	a	129	ü	161	í	193	┴	225	ß
2	☻	34	"	66	B	98	b	130	é	162	ó	194	┬	226	Γ
3	♥	35	#	67	C	99	c	131	â	163	ú	195	├	227	π
4	♦	36	$	68	D	100	d	132	ä	164	ñ	196	─	228	Σ
5	♣	37	%	69	E	101	e	133	à	165	Ñ	197	┼	229	σ
6	♠	38	&	70	F	102	f	134	å	166	ª	198	╞	230	µ
7	•	39	'	71	G	103	g	135	ç	167	º	199	╟	231	τ
8	◘	40	(	72	H	104	h	136	ê	168	¿	200	╚	232	Φ
9	○	41	)	73	I	105	i	137	ë	169	⌐	201	╔	233	Θ
10	◙	42	*	74	J	106	j	138	è	170	¬	202	╩	234	Ω
11	♂	43	+	75	K	107	k	139	ï	171	½	203	╦	235	δ
12	♀	44	,	76	L	108	l	140	î	172	¼	204	╠	236	∞
13	♪	45	-	77	M	109	m	141	ì	173	¡	205	═	237	ø
14	♫	46	.	78	N	110	n	142	Ä	174	«	206	╬	238	ε
15	☼	47	/	79	O	111	o	143	Å	175	»	207	╧	239	∩
16	►	48	0	80	P	112	p	144	É	176	░	208	╨	240	≡
17	◄	49	1	81	Q	113	q	145	æ	177	▒	209	╤	241	±
18	↕	50	2	82	R	114	r	146	Æ	178	▓	210	╥	242	≥
19	‼	51	3	83	S	115	s	147	ô	179	│	211	╙	243	≤
20	¶	52	4	84	T	116	t	148	ö	180	┤	212	╘	244	⌠
21	§	53	5	85	U	117	u	149	ò	181	╡	213	╒	245	⌡
22	▬	54	6	86	V	118	v	150	û	182	╢	214	╓	246	÷
23	↨	55	7	87	W	119	w	151	ù	183	╖	215	╫	247	≈
24	↑	56	8	88	X	120	x	152	ÿ	184	╕	216	╪	248	°
25	↓	57	9	89	Y	121	y	153	Ö	185	╣	217	┘	249	∙
26	→	58	:	90	Z	122	z	154	Ü	186	║	218	┌	250	·
27	←	59	;	91	[	123	{	155	¢	187	╗	219	█	251	√
28	∟	60	<	92	\	124	¦	156	£	188	╝	220	▄	252	ⁿ
29	↔	61	=	93	]	125	}	157	¥	189	╜	221	▌	253	²
30	▲	62	>	94	^	126	~	158	₧	190	╛	222	▐	254	■
31	▼	63	?	95	_	127	⌂	159	ƒ	191	┐	223	▀	255	