TV: TeleVisión – Plan 2010

Codificación estadística

Contenido

- ★ Definiciones
  - ✸ Código, código no singular, instantáneo, decodificabilidad
  - ✸ Inecuación de Kraft-McMillan
  - ✸ Longitud media y rendimiento de un código
- ★ Códigos Huffman
- ★ Códigos subóptimos
- ★ Codificación de longitud de recorridos
- ★ Codificación aritmética
- ★ Códigos basados en diccionarios

Código

Dada una variable aleatoria X, un código fuente para dicha variable aleatoria es una aplicación entre $\mathcal{X}$, el rango de valores de X (alfabeto fuente) y $\mathcal{D}^+$, el conjunto de las cadenas de símbolos de longitud finita de un alfabeto de D símbolos (alfabeto código).

C(x) es la palabra código correspondiente a x y l(x) representa la longitud de dicha palabra código.

Por ejemplo, si C(alto)=00 y C(bajo)=11 es un código fuente para $\mathcal{X}$={alto,bajo} con el alfabeto $\mathcal{D}$={0,1}

Un **código** asocia a cada símbolo del alfabeto fuente una secuencia fija de símbolos del alfabeto código.

Código no singular

Un código se denomina **no singular** si a cada elemento de X le corresponde una palabra código diferente, esto es

$$\forall x, x' \in \mathcal{X},\ x \neq x' \quad \Rightarrow \quad C(x) \neq C(x')$$

Ejemplo. Para $\mathcal{X}$={a,b,c,d}

Símbolo Fuente	Código
a	0
b	1
c	00
d	11

Extensión de un código

Definición: La extensión de orden p de la fuente X con n símbolos $x_1, x_2, \cdots x_n$ es la fuente X^p con n^p símbolos, donde cada símbolo de X^p puede considerarse una concatenación diferente de p símbolos de X.

La extensión C^n de un código es una aplicación que hace corresponder secuencias de longitud n del alfabeto fuente (símbolos de X^p) con secuencias de longitud finita de símbolos del alfabeto código:

$$C^n(x_1 x_2 \cdots x_n) = C(x_1)C(x_2)\cdots C(x_n)$$

La extensión C^* de un código C es una aplicación que hace corresponder secuencias de longitud arbitraria del alfabeto fuente con secuencias de longitud finita de símbolos del alfabeto código:

$$C^* = C \cup C^2 \cup \ldots \cup C^n \cup \ldots = \bigcup_{i=1}^{\infty} C^i$$

La extensión de orden 2 del código anterior es:

aa	00	ba	10	ca	000	da	110
ab	01	bb	11	cb	001	db	111
ac	000	bc	100	cc	0000	dc	1100
ad	011	bd	111	cd	0011	dd	1111

Código unívocamente decodificable

Un código se dice que es unívocamente decodificable si cada secuencia finita de símbolos del alfabeto código corresponde como máximo a un mensaje.

Puede demostrarse que un código bloque es unívocamente decodificable si y solo si su extensión C* es no singular.

Para el ejemplo anterior:

abcd	010011
abaabb	010011

La decodificación (esto es, determinar la secuencia de símbolos fuente que han dado lugar a una secuencia código) no es posible de forma unívoca porque existe más de una forma de obtener la misma secuencia código.

El código del ejemplo no es unívocamente decodificable.

Código instantáneo o prefijo

Un código unívocamente decodificable es un código instantáneo si es posible resolver cada uno de los símbolos de la secuencia sin el concurso de los símbolos que le suceden en la misma.

Ejemplo: Para los códigos unívocamente decodificables:

Fuente	C-1	C-2	C-3
A	00	0	0
B	01	10	01
C	10	110	011
D	11	1110	0111

Las secuencias correspondientes a la codificación de la cadena "abc" son:

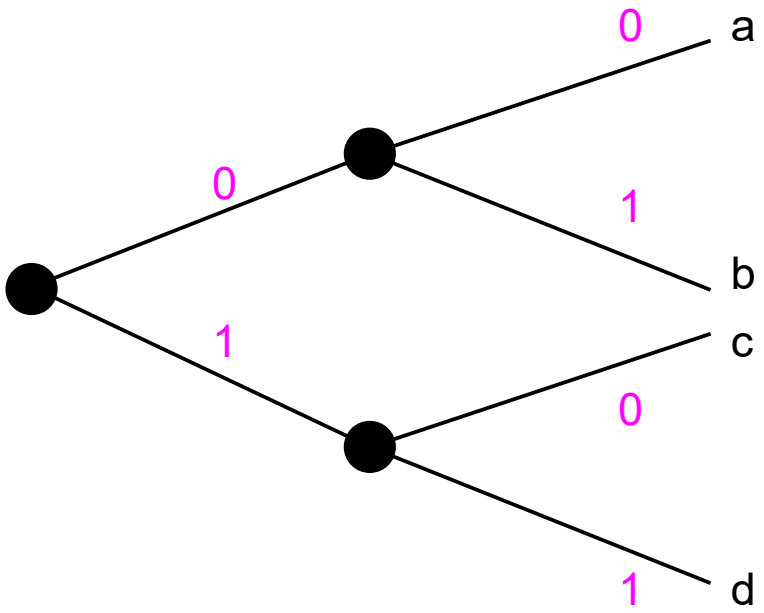
Código	Secuencia
C-1	000110
C-2	010110
C-3	001011

C-1 y C-2 son instantáneos. Sin embargo, para C-3 una vez recibido el primer 0, es necesario examinar el siguiente símbolo (0) para poder realizar la decodificación (código no instantáneo).

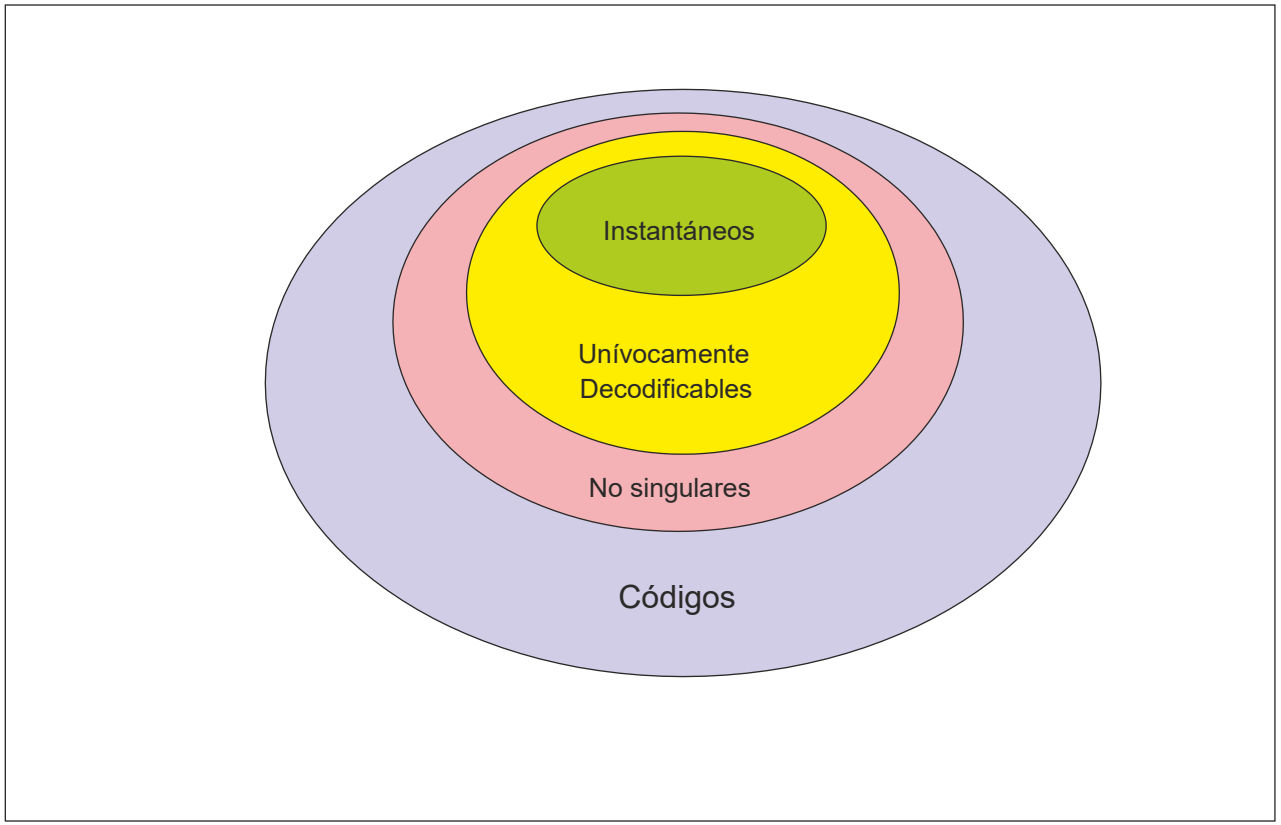
Código instantáneo

La condición necesaria y suficiente para que un código sea instantáneo es que ninguna palabra código coincida con el prefijo de otra palabra código.

Los códigos instantaneos pueden decodificarse facilmente con ayuda de un arbol de decisión. Para ello basta disponer las palabras del código en forma de arbol, etiquetando cada rama de éste con uno de los símbolos correspondientes al alfabeto. No tienen por que emplearse todas las posibles ramas del arbol, pero todas las ramas deben acabar en palabras código válidas. Las ramas que faltan indican palabras código no asignadas. Por ejemplo, el árbol de decodificación correspondiente al código C-1 anterior es:



Clasificación de los códigos



Inecuación de Kraft-McMillan

Sea un alfabeto fuente $S = \{x_1, x_2, \dots, x_m\}$ y un alfabeto código $D = \{d_1, d_2, \dots, d_r\}$ con el que se desea construir un código instantáneo para dicha fuente. Puede demostrarse que la condición necesaria y suficiente para la existencia de un código instantáneo de longitudes $l_1, l_2, \dots l_m$ es que:

$$\sum_{i=1}^{m} r^{-l_i} \leq 1$$

En el caso r=2 (binario), la inecuación de Kraft-McMillan se transforma en:

$$\sum_{i=1}^{m} 2^{-l_i} \leq 1$$

Entropia

Sea S una fuente que emite símbolos los símbolos $\{x_1,\ldots,x_n\}$ de forma independiente entre sí (esto es, de forma que $\forall i,j \quad p(x_i x_j) = p(x_i)p(x_j)$).

Se define la entropía de la fuente, $H(S)$ de la forma:

$$H(S) = -\sum_{i=1}^{n} p(x_i)\, log_b(p(x_i))$$

Habitualmente la base del logaritmo es b=2 y entonces la entropía se mide en bits.

Longitud media de un código

Sea C un código que asocia a cada $x_i \in \mathcal{X}$, una palabra código $C(x_i)$. Sea $l(x_i)$ la longitud de cada una de esas palabras código y $p(x_i)$ la probabilidad del símbolo x_i. Se define la longitud media del código, L, mediante la ecuación:

$$L = \sum_i p(x_i) l(x_i)$$

Se dice que un código basado en un determinado alfabeto código es compacto respecto a una determinada distribución de probabilidades si no puede construirse un código basado en dicho alfabeto con longitud media menor.

Puede demostrarse (Primer teorema de Shannon) que, si L_n representa la longitud media obtenida al codificar la extensión de orden n de la fuente S con un código compacto:

$$\lim_{n\to\infty} \frac{L_n}{n} = H(S)$$

Rendimiento de un código

Se define el rendimiento de un código, η, como

$$\eta = \frac{H(S)}{L}$$

…donde L representa la longitud media del código

- El rendimiento es un valor positivo menor o igual a la unidad.
- El complemento a 1 del rendimiento se denomina redundancia.
- Para conseguir códigos eficientes se requiere que la longitud media sea lo menor posible, esto es, que se asignen palabras código de menor longitud a los símbolos más probables.

Codificación

Códigos óptimos (Huffman)

Código Óptimo

La longitud media de un código depende de la longitud de cada palabra código:

$$L = \sum_i p(x_i) l(x_i)$$

Un código es tanto mejor cuanto menor sea su longitud media. Para ello, una buena idea es asignar las palabras código de menor longitud a los símbolos más probables.

Un código es óptimo (o compacto) si no existe otro código con longitud media menor.

En 1952 Huffman presentó un algoritmo para construir códigos instantáneos óptimos.

Códigos Huffman

Aquí presentaremos como construir códigos binarios, aunque el mecanismo es básicamente el mismo para códigos r-arios.

El algoritmo de codificación puede ilustrarse fácilmente como el proceso de construcción de un árbol de decodificación partiendo de sus hojas:

Algoritmo

1. Sea L la lista de probabilidades de los símbolos del alfabeto fuente. Cada uno de estos símbolos (con su probabilidad) constituyen las hojas del árbol.

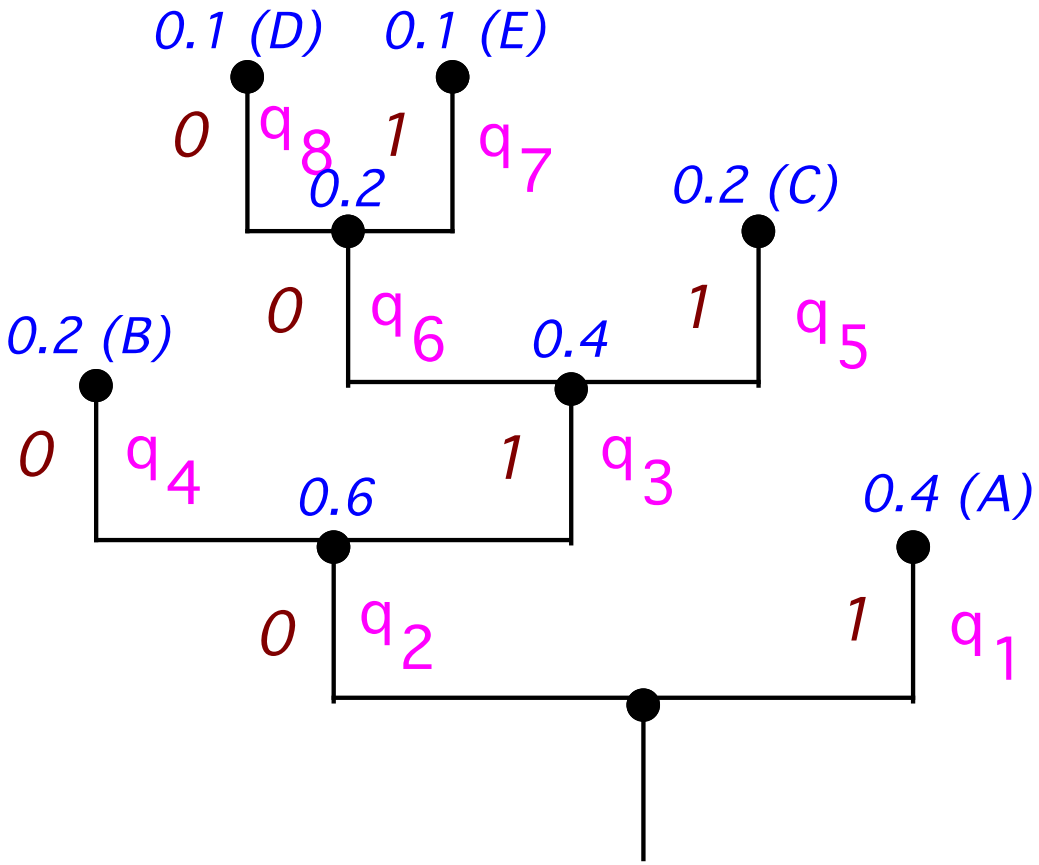
2. Tomar las dos probabilidades más pequeñas de la lista, generar un nodo intermedio como su nodo padre, etiquetando una rama saliente del nodo padre como 0 y la otra como 1.

3. Substituir los dos nodos anteriores por el nodo padre recién formado, etiquetándolo con la suma de las probabilidades anteriores. El número de nodos en la lista se habrá reducido en una unidad. Si el número de nodos obtenido es la unidad, se ha obtenido el arbol de decodificación (y el correspondiente código Huffman). En caso contrario, volver al paso 2.

Códigos Huffman: ejemplo

Construcción de un código Huffman para un la fuente S que emite los símbolos {A,B,C,D,E} con probabilidades 0.4, 0.2, 0.2, 0.1 y 0.1

El código binario se forma recorriendo desde la raíz hasta cada hoja el árbol de decodificación.



Eficiencia de los códigos Huffman

La longitud media de un código óptimo (como Huffman) para una fuente S verifica:

$$H(S) \le \overline{L} \le H(S)+1$$

Si codificamos extensiones de orden n de la fuente la ecuación anterior se transforma en:

$$H(S^n) \le \overline{L} \le H(S^n)+1$$

Llamando $\overline{L}_n = \dfrac{\overline{L}}{n}$ a la longitud media por símbolo fuente:

$$\frac{H(S^n)}{n} \le \overline{L}_n \le \frac{H(S^n)}{n} + \frac{1}{n}$$

Si $p(x_1 x_2 \ldots x_n) = p(x_1)\,p(x_2)\ldots p(x_n)$ (independencia) $H(S^n) = nH(S)$ y

$$H(S) \le \overline{L}_n \le H(S) + \frac{1}{n}$$

Eficiencia de los códigos Huffman

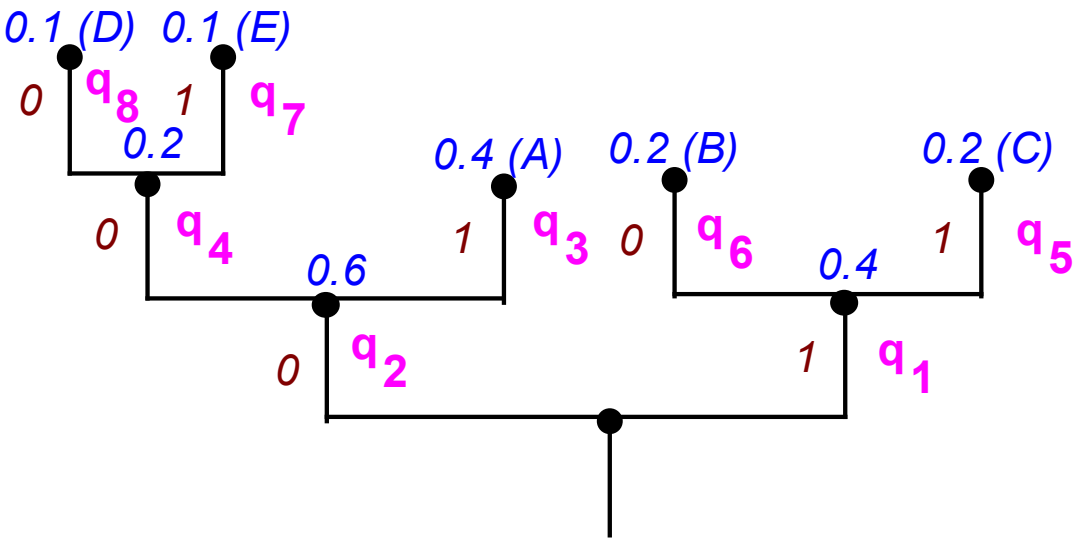
Se conocen mejores cotas. Si p es la probabilidad del símbolo más probable y $\bar{L}$ es la longitud media del código Huffman,

$$\bar{L} \leq \begin{cases} H(S) + p_{\max} & \text{Si } p \geq 0.5 \\ H(S) + p_{\max} + 0.086 & \text{Si } p < 0.5 \end{cases}$$

donde se ha aproximado $1 - \log_2 e + \log_2(\log_2 e) \approx 0.086$

Códigos Huffman de mínima varianza

Para la construcción del segundo código se ha modificado el paso 2 algoritmo, de manera que, siempre que existan múltiples alternativas en la elección de los dos nodos se tomarán aquellos que sean raíces de árboles de menor profundidad. Con esta estrategia se obtiene un árbol más balanceado. Siguiendo esta estrategia se obtienen códigos Huffman de mínima varianza.



Códigos Huffman de mínima varianza

La elección de los dos elementos de menor probabilidad durante el paso 2 del algoritmo de construcción de un código Huffman puede no ser única.

La varianza de las longitudes de la palabra código se define como:

$$V(l) = \sum_{i=1}^{N} \left(l_i - E[l] \right)^2 p_i$$

A igualdad de longitud media son preferibles los códigos de menor varianza porque las palabras código tenderán a tener longitudes similares, más próximas a la longitud media, y cabe esperar menor variabilidad en la longitud del código de mensajes de longitud finita. Si comparamos la varianza de las longitudes de las palabras código correspondientes a los dos códigos construidos obtenemos:

Código no. 1:

$$V(l) = ((1-2.2)^2 \times 0.4) + ((2-2.2)^2 \times 0.2) + ((3-2.2)^2 \times 0.2) + ((4-2.2)^2 \times 0.2) = 1.36$$

Código no. 2:

$$V(l) = ((2-2.2)^2 \times 0.8) + ((3-2.2)^2 \times 0.1) + ((3-2.2)^2 \times 0.2) = 0.16$$

Ejemplo

Una fuente que emite los símbolos 0 y 1 con probabilidades 0.7 y 0.3 respectivamente, de manera independiente entre si.

1. Calcular la entropía de la fuente
2. Calcular un código Huffman de mínima varianza para la extensión de orden 3 de esta fuente.
3. Calcular la eficiencia del código obtenido.

Solución (1)

1. La extensión de orden tres se construye fácilmente calculando la probabilidad de cada combinación de tres símbolos:

Sec.		Prob.	-p*log$_2$(p)
000	$(0,7)^3$	0,343	0,5295
001	$(0,7)^2(0,3)$	0,147	0,4066
010	$(0,7)^2(0,3)$	0,147	0,4066
100	$(0,7)^2(0,3)$	0,147	0,4066
011	$(0,7)\ (0,3)^2$	0,063	0,2513
101	$(0,7)\ (0,3)^2$	0,063	0,2513
110	$(0,7)\ (0,3)^2$	0,063	0,2513
111	$(0,3)^3$	0,027	0,1407
			2,6439

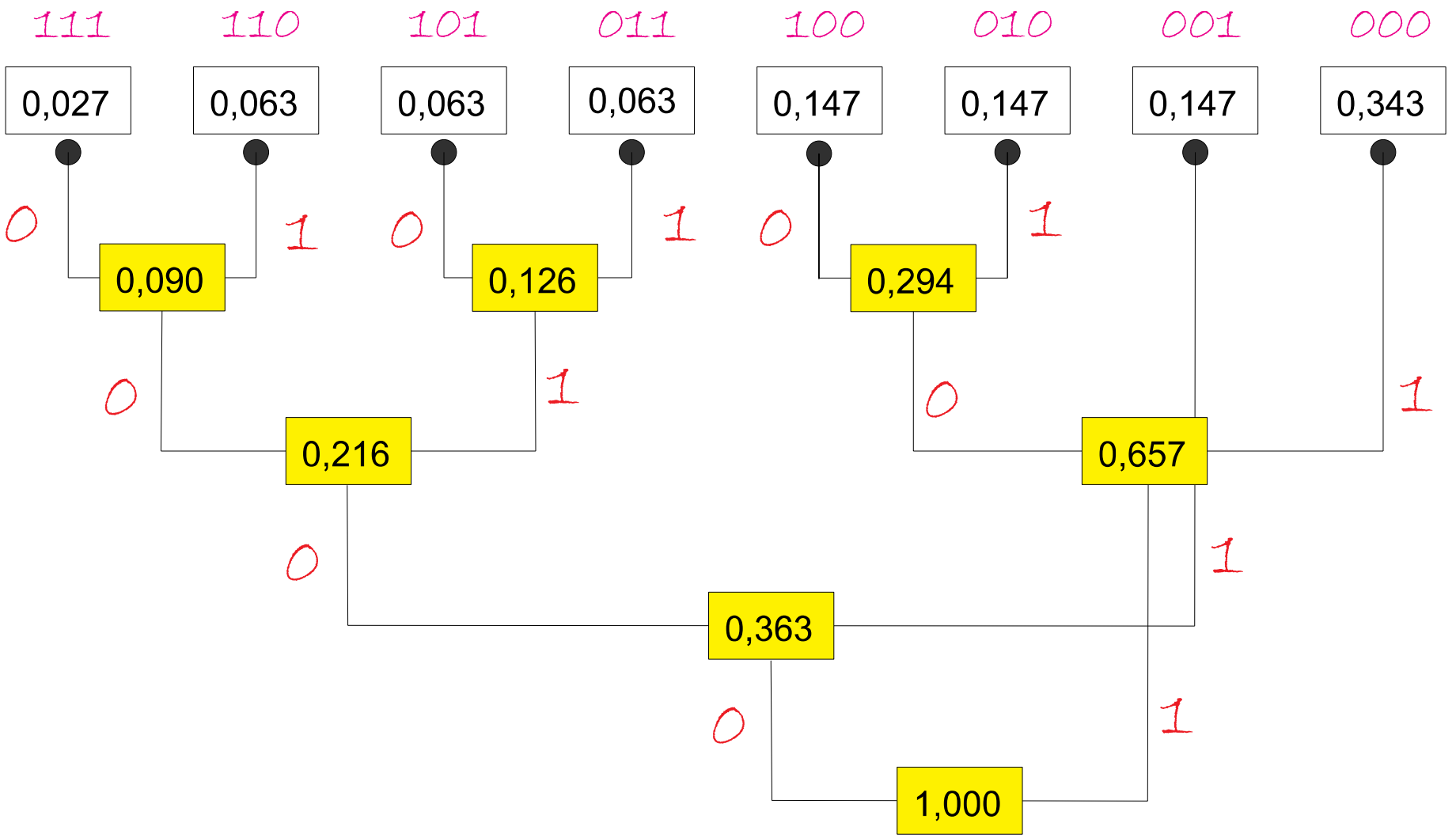
Entropía:

$$-\sum_i p_i \cdot \log_2 p_i = 2.6439$$

2. La construcción del código Huffman se ilustra en el gráfico siguiente, obteniéndose los códigos de la tabla adjunta (la solución no es única).

Sec.	Código
000	11
001	01
010	101
100	100
011	0011
101	0010
110	0001
111	0000

Solución (2)


111
110
101
011
100
010
001
000
0,027
0,063
0,063
0,063
0,147
0,147
0,147
0,343
0
1
0,090
0,126
0,294
0,216
0,657
0,363
1,000

Solución (3)

3. Longitud media del código

$$\bar{L} = \sum_i l_i \cdot p_i = 2.726$$

$$\eta = \frac{H(S)}{\bar{L}} = 0.9698...$$

Sec.	Prob.	Código	Longitud	lxp
000	0,343	11	2	0,686
001	0,147	01	2	0,294
010	0,147	101	3	0,441
100	0,147	100	3	0,441
011	0,063	0011	4	0,252
101	0,063	0010	4	0,252
110	0,063	0001	4	0,252
111	0,027	0000	4	0,108
				2,726

Códigos Huffman adaptativos

¿Por qué un código Huffman adaptativo?

- Se puede construir un código Huffman adaptativo haciendo que transmisor y receptor contabilicen (y construyan) el árbol de probabilidades simultáneamente.
- El código inicial es cualquier representación previamente acordada (por ejemplo un código bloque)
- Para codificar un símbolo
- se busca dicho símbolo en el árbol.
  - Si está presente, se asigna la palabra código correspondiente recorriendo desde la hoja hacia la raíz, asignando 0 o 1 según corresponda, y se incrementa en una unidad la cuenta correspondiente a ese símbolo.
  - Si el símbolo no está presente, se transmite un símbolo especial, NYT, seguido del código correspondiente a dicho símbolo en el código inicial, y se incluye dicho símbolo en el árbol con cuenta 1. El árbol se rebalancea si es necesario para que siga correspondiendo a un código Huffman.
- El símbolo NYT está en el árbol, siempre con probabilidad 0.

Codificación

Códigos sub-óptimos

Códigos sub-óptimos

- (Posibles) razones para su uso
  - Número de símbolos infinito (p.e. Números racionales)
  - Diccionarios demasiado grandes
  - Requisitos de simplicidad
  - Etc…

Código Huffman modificado

- En ocasiones el número de símbolos hace que la aplicación de un código Huffman no sea práctica (por ejemplo, si algunas palabras código pueden tener cientos de bits)
- En ese caso puede construirse un código modificado de la siguiente forma:
  - Agrupar los símbolos con menor probabilidad e identificarlos con un símbolo específico (ESCAPE)
  - Construir un código Huffman para los símbolos restantes más el ESCAPE
  - Codificar los símbolos del grupo de menor probabilidad como la concatenación del código Huffman correspondiente al ESCAPE más un código binario de longitud fija.

Ejemplo

Una fuente que emite los símbolos 0 y 1 con probabilidades 0.7 y 0.3 respectivamente, de manera independiente entre si. Diseñar un código Huffman modificado para la extensión de orden 6 de dicha fuente.

Solución (1)

En una secuencia de n dígitos binarios, el número de las que tiene exactamente k ceros es $\binom{n}{k}$ y su probabilidad es $p^k(1-p)^{n-k}$

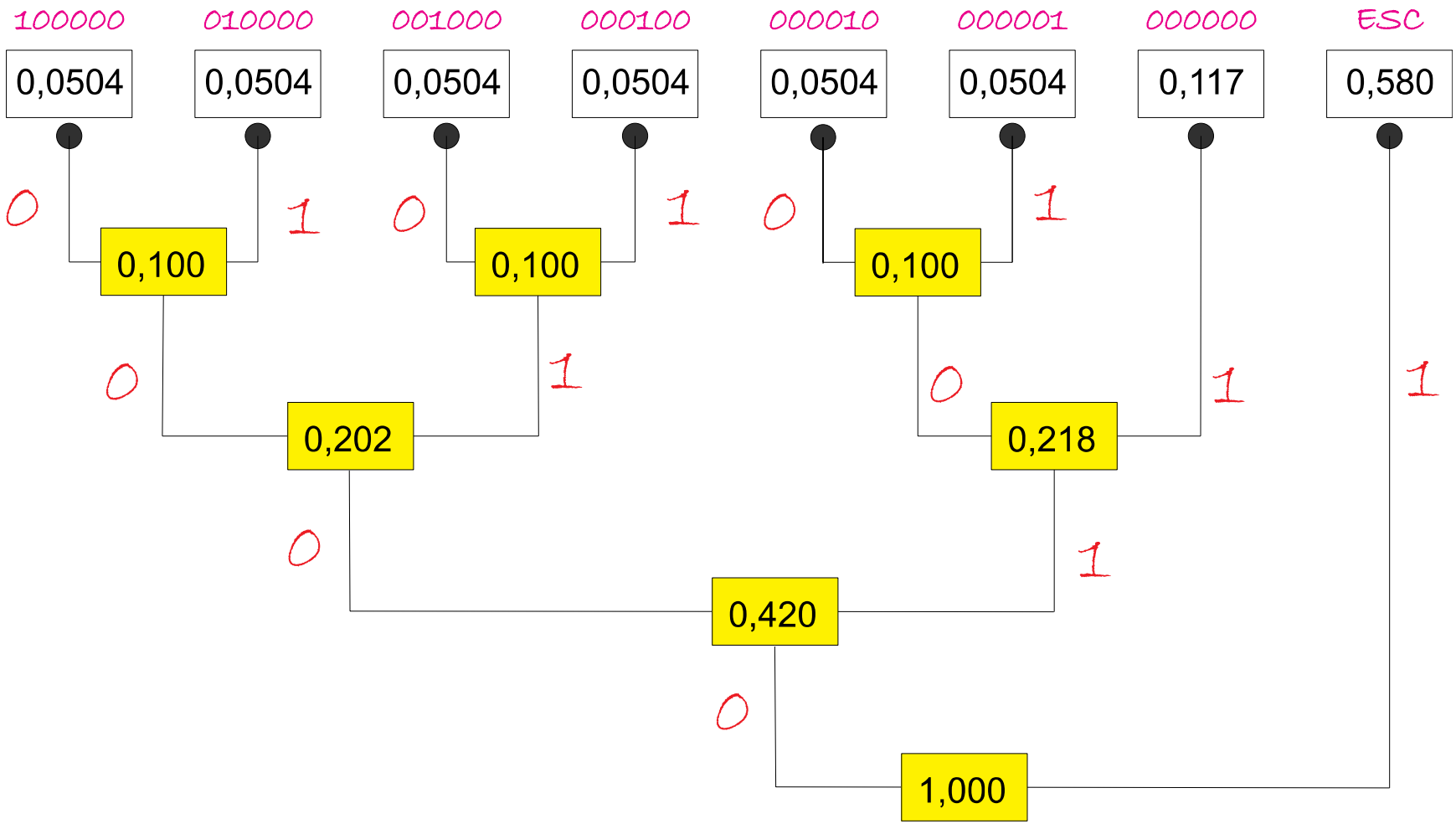
Las secuencias más probables son

1. La que tiene 6 ceros, con probabilidad 0.7^6
2. Las 6 que tienen un uno, con probabilidades $0.7^5(0.3)^1$
3. El "escape", con probabilidad $1-0.7^5(0.3)^1-0.7^6$

Secuencia	Probabilidad
000000	0,117649
000001	0,050421
000010	0,050421
000100	0,050421

Secuencia	Probabilidad
001000	0,050421
010000	0,050421
100000	0,050421
ESC	0,579825

Solución (2)


100000
010000
001000
000100
000010
000001
000000
ESC
0,0504
0,0504
0,0504
0,0504
0,0504
0,0504
0,117
0,580
0
1
0,100
0,100
0,100
0
1
0,202
0,218
0
1
0,420
0
1
1,000

Solución (3)

Las cadenas con más de un uno se codifican concatenando la secuencia al código de ESCAPE.

Ejemplos:

000000 ➔ 011
000001 ➔ 0101
100000 ➔ 0000
100001 ➔ 1100001
111111 ➔ 1111111

Código SHIFT

Un código shift k se construye a partir de las 2^k palabras código de idéntica longitud que pueden construirse con k bits. De éstas, a m palabras código se asigna el significado de shift o desplazamiento. Las restantes 2^k-m palabras código (terminales) se asignan a los símbolos más probables. Los símbolos menos probables se codifican precediendo un terminal de combinaciones de una o más palabras de shift.

Símbolo	Prob.	Código S2	lxp	Código S1	lxp
A	0.8	01	1.6	1	0.8
B	0.1	10	0.2	01	0.4
C	0.05	11	0.1	001	0.15
D	0.02	0001	0.08	0001	0.08
E	0.01	0010	0.04	00001	0.05
F	0.01	0011	0.04	000001	0.06
G	0.005	000001	0.04	0000001	0.035
H	0.005	000010	0.04	0000000	0.035
Total	1		2.14		1.41

Códigos B

Los códigos B son código cuasi óptimos para distribuciones de probabilidades de la forma $p_k = k^{-\gamma}$.

Cada palabra código de un código Bn se compone de un número entero de bloques de (n+1) bits. Cada bloque está compuesto de n bits de información y un bit de continuación. El bit de continuación indica si el bloque que controla forma parte de la palabra código que se está de codificando o de la siguiente.

Símbolo	Prob.	Código B1	lxp	Código B2	lxp
A	0.8	C0	1.6	C00	2.4
B	0.1	C1	0.2	C01	0.6
C	0.05	C0C0	0.2	C10	0.15
D	0.02	C0C1	0.08	C11	0.06
E	0.01	C1C0	0.04	C00C00	0.06
F	0.01	C1C1	0.04	C00C01	0.06
G	0.005	C0C0C0	0.03	C00C10	0.03
H	0.005	C0C0C1	0.03	C00C11	0.03
Total	1		2.22		3.39

Códigos B

La forma en la que el bit de continuación señala las palabras código son variadas.

Por ejemplo, si el bit de continuación alterna su valor entre palabras código consecutivas, ABCDE se codificaría usando el código B1 del ejemplo como:

C0C1C0C0C0C1C1C0 → 00.11.0000.1011.0100

Con el código B2 del ejemplo, si el bit de continuación a 1 indica fin de palabra código.

C00C01C10C11C00C00 → 100.101.110.111.000100

Representación de enteros: código unario

- ★ El código unario para un entero positivo, n, es, simplemente, n unos seguidos de un cero.
- ★ Es óptimo si $P[k]=2^{-k}$
- ★ No es necesario que el número de símbolos este acotado.
- ★ Es básicamente un código S1, donde el “SHIFT” se codifica con 1

entero	Código
0	0
1	10
4	11110
7	11111110

Código de Golomb

Familia de códigos diseñado para codificar secuencias de números enteros en los que la probabilidad del entero decrece con su valor. La familia se caracteriza por un entero, m.

Un entero k, se codifica como la pareja (q, r) donde $q = \left\lfloor \frac{k}{m} \right\rfloor$ y $r = k \bmod m$.

- El primer elemento de la pareja, q, se codifica usando un código unario, es decir, q unos seguidos de un cero.
- Para el segundo elemento se usa una representación binaria de r.

Es óptima si $P(n) = p^{n-1}(1-p)$ con $m = \left\lceil -\frac{1}{\log_2 p} \right\rceil$

Ejemplo (m=8)

k	q	r	q	r	código
50	6	2	1111110	010	1111110010
5	0	5	0	101	0101
0	0	0	0	000	0000

Código de Golomb: representación de enteros

Para mejorar la eficiencia se optimiza la representación binaria de r como sigue:

- Si m es potencia de 2, la representación binaria de r
- Si m no es potencia de 2, palabras código de longitud $\lfloor \log_2 m \rfloor$ para los $2^{\lceil \log_2 m \rceil} - m$ primeros valores y la representación binaria de $r + 2^{\lceil \log_2 m \rceil} - m$ (palabras código de longitud $\lceil \log_2 m \rceil$) para el resto de valores.

Ejemplo m=6, $r \in \{0,1,2,3,4,5\}$

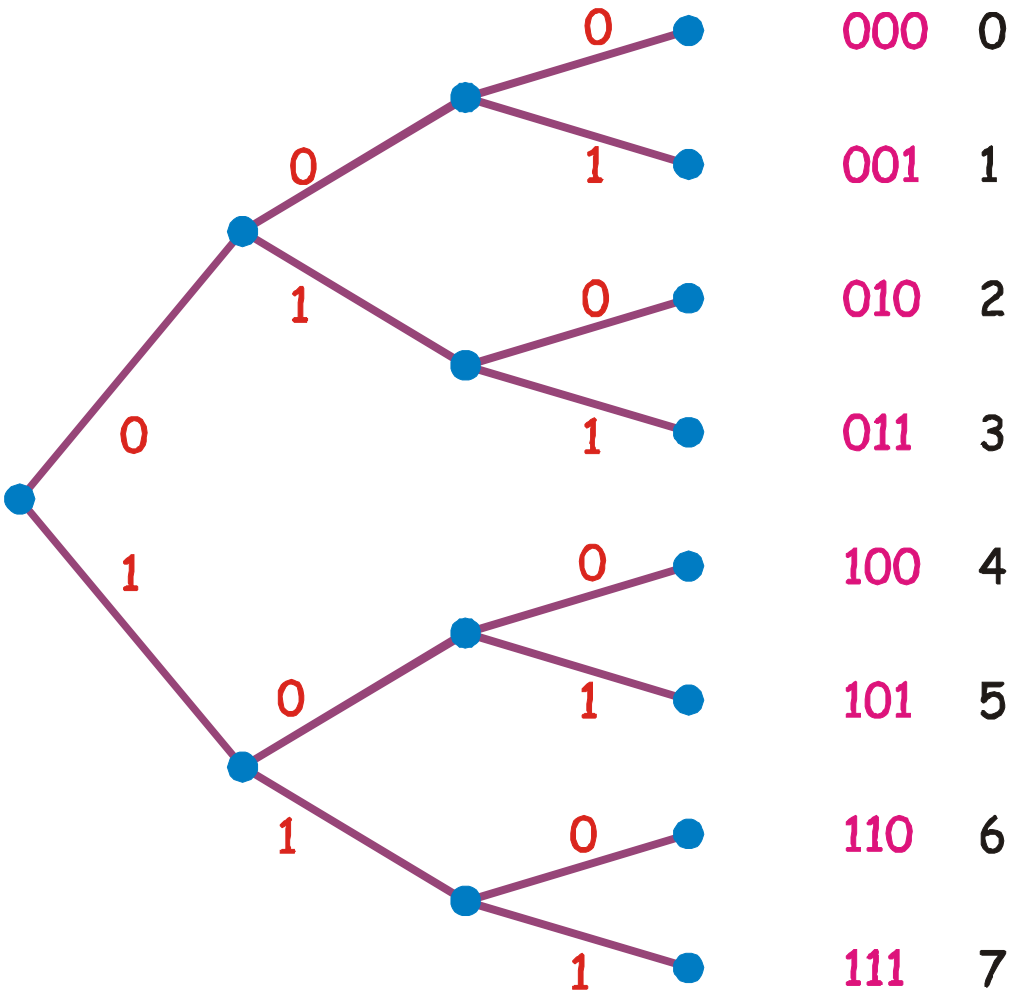
$\log_2 6 = 2{,}585$, $\lfloor \log_2 6 \rfloor = 2$, $\lceil \log_2 6 \rceil = 3$

los primeros $2^{\lceil \log_2 m \rceil} - m = 2^3 - 6 = 2$ valores se representan con 2 bits

los 4 restantes con la representación binaria de $r + 2^3 - 6 = r + 2$ (esto es, de 4,5,6 y 7)

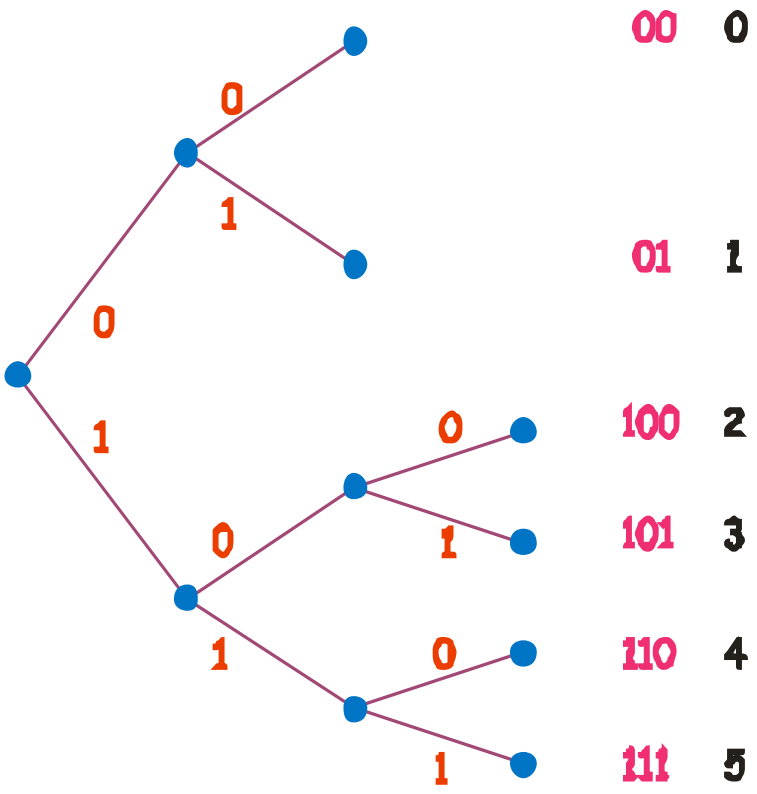
0	00	3	101
1	01	4	110
2	100	5	111

Representación de enteros (0 a m-1)



- ★ $P(0) \geq P(1) \geq .. \geq P(m-1)$
- ★ Si $m=2^k$, códigos de k bits por símbolo
- ★ Ejemplo: m=8, k=3

Representación de enteros (0 a m-1)



- ★ $P(0) \geq P(1) \geq .. \geq P(m-1)$
- ★ Si $m \neq 2^k$, puede "podarse el árbol"
- ★ Cada rama que se poda reduce en una unidad el número de hojas del árbol
- ★ Podamos por arriba para asignar las cadenas más cortas a los símbolos más probables
- ★ Ejemplo: m=6, han de podarse $2^3-6=2$ ramas

Código de Golomb exponencial (Exp-Golomb)

- Con 1 bit podemos representar dos enteros (p.e. 1 y 2), con 2 bits podemos representar cuatro, con 3 bits 8 y así sucesivamente
- La idea del código de Golomb es representar los números enteros por la pareja (bits, número)
- El número de parejas de que disponemos es infinito:
  - (1,0) (1,1)
  - (2,00), (2,01), (2,10), (2,11)
  - (3,000), (3,001), (3,010), (3,011), (3,000), (3,001), (3,010), (3,011)
- El número de bits se representa por un código unario.

		100	1	11000	3	1110000	7
		101	2	11001	4	1110001	8
				11010	5	1110010	9
				11011	6	1110011	10
						1110100	11
						1110101	12
						1110110	13
						1110111	14

- Hay una combinacion que no hemos usado. ¿Cuál?

Código de Golomb exponencial (Exp-Golomb)

- ★ Sirve para codifica números enteros no negativos.
- ★ Es eficiente si la probabilidad de un entero decrece con su magnitud.
- ★ El número de enteros tales que $2^n-1 \leq m < 2^{n+1}-1$ es exactamente 2^n, que por lo tanto pueden representarse con n bits.
- ★ El primer entero de cada intervalo es 2^n-1
- ★ Cada entero del intervalo, m, puede codificarse con el código binario de n bits correspondiente a $m-(2^n-1)=m-2^n+1$
- ★ El intervalo se codifica mediante un código “unario”.

n	2^n-1	$2^{n+1}-1$	Enteros en el intervalo
0	0	1	0
1	1	3	1,2
2	3	7	3,4,5,6
3	7	15	7,8,9,10,11,12,13,14

Código de Golomb exponencial (Exp-Golomb)

Para codificar el número entero no negativo k:

- El intervalo es: $n = \lfloor \log_2(k+1) \rfloor$, y se representa por n unos seguidos de un cero (o n ceros seguidos de un 1)
- Se codifica $k - 2^n + 1$ usando un código bloque de n bits

El número total de bits utilizado es 2n+1

k	Código	k	Código	k	Código
0	*0*	4	*11001*	8	*1110001*
1	*100*	5	*11010*	9	*1110010*
2	*101*	6	*11011*	10	*1110011*
3	*11000*	7	*1110000*	11	*1110100*

Codificación

Codificación de longitud de recorridos

Codificación de longitud de recorridos

- En la codificación de longitud de recorridos en la que, en vez de codificar cada símbolo independientemente se codifica la longitud de las secuencias con un valor homogéneo.
- Ejemplo: en vez de codificar

Como:

```
NNNBBBBBBBBBBBBBBBNNNNNNNNNNBBBBBBBBBBBBBBBBBBBB
```

Se codifica:

```
3N15B10N22B
```

O mejor aún, y dado que las secuencias de blancos y negros deben alternarse, suponiendo que empezamos siempre en una secuencia de blancos:

```
0 3 15 10 22
```

¿Por qué es mejor que codificar cada símbolo?

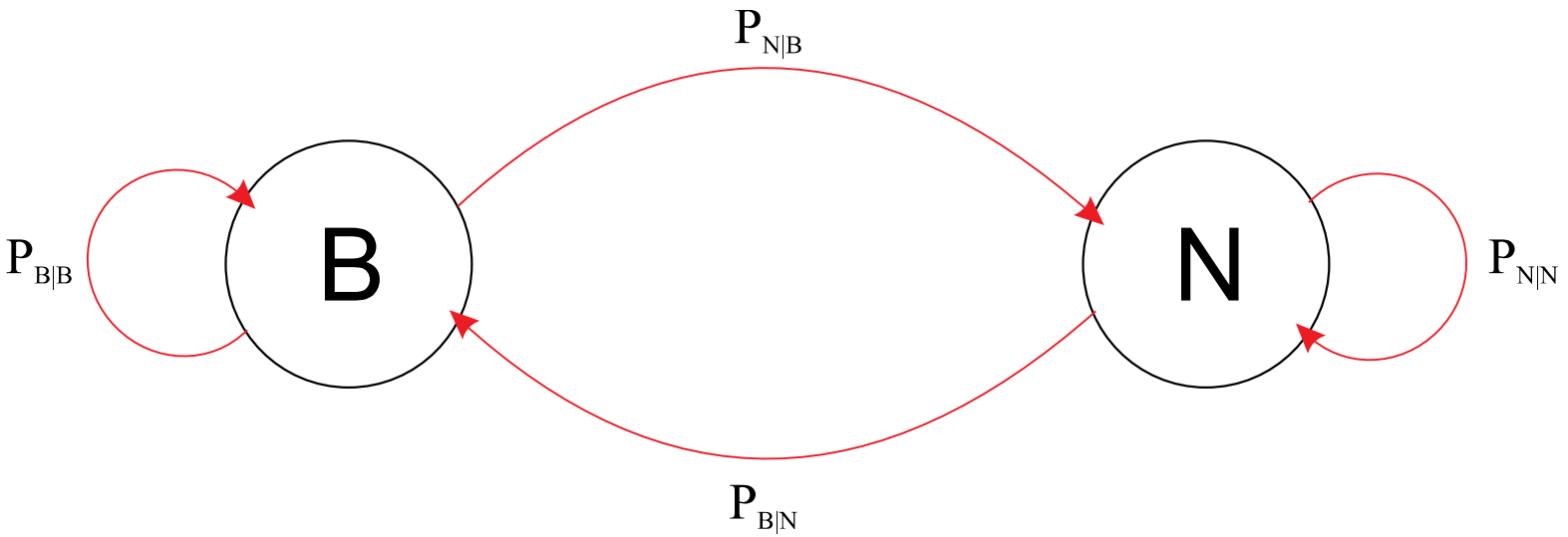
Ejemplo: Codificación de longitud de recorridos

Modelo de Markov para una imagen binaria:

★Dos estados (Blanco y Negro) con probabilidades P_B y P_N de transición $P_{N|B}$, $P_{B|B}$, $P_{B|N}$, $P_{N|N}$

★P_N representa la probabilidad de que un pixel tomado al azar sea negro

★$P_{N|B}$ (por ejemplo) representa la probabilidad de que el siguiente píxel sea negro si el actual es blanco.



Ejemplo: Codificación de longitud de recorridos

La probabilidad de que el píxel i+1 sea Blanco es $P_B^{i+1} = P_B^i \cdot P_{B|B} + P_N^i \cdot P_{B|N}$

y la de que sea Negro $P_N^{i+1} = P_N^i \cdot P_{N|N} + P_B^i \cdot P_{N|B}$

En forma matricial:

$$\begin{bmatrix} P_B^{i+1} \\ P_N^{i+1} \end{bmatrix} = \begin{bmatrix} P_{B|B} & P_{B|N} \\ P_{N|B} & P_{N|N} \end{bmatrix} \begin{bmatrix} P_B^i \\ P_N^i \end{bmatrix}$$

En régimen estacionario $P_B^{i+1} = P_B^i$ y $P_N^{i+1} = P_N^i$ por lo que :

$$\begin{bmatrix} P_B \\ P_N \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} P_B \\ P_N \end{bmatrix} = \begin{bmatrix} P_{B|B} & P_{B|N} \\ P_{N|B} & P_{N|N} \end{bmatrix} \begin{bmatrix} P_B \\ P_N \end{bmatrix}$$

$$\begin{bmatrix} P_{B|B} - 1 & P_{B|N} \\ P_{N|B} & P_{N|N} - 1 \end{bmatrix} \begin{bmatrix} P_B \\ P_N \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \text{ junto con } P_B + P_N = 1$$

Ejemplo (1 de 2)

Supongamos $P_{B|B} = 0.99 \quad P_{N|N} = 0.7$.

Debe ser $P_{N|B} = 0.01 \quad P_{N|N} = 0.7$ con lo que :

$$\begin{bmatrix} 0.99-1 & 0.3 \\ 0.01 & 0.7-1 \end{bmatrix} \begin{bmatrix} P_B \\ P_N \end{bmatrix} = \begin{bmatrix} -0.01 \cdot P_B + 0.3 \cdot P_N \\ 0.01 \cdot P_B - 0.3 \cdot P_N \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

$$\left.\begin{array}{l} -0.01 \cdot P_B + 0.3 \cdot P_N = 0 \\ P_B + P_N = 1 \end{array}\right\} \rightarrow -0.01 \cdot P_B + 0.3 \cdot (1 - P_B) = 0 \rightarrow 0.31 \cdot P_B = 0.3$$

$$P_B = \frac{30}{31} \quad P_N = \frac{1}{31}$$

Si los considerásemos símbolos independientes:

$$H = -\frac{30}{31}\log_2\frac{30}{31} - \frac{1}{31}\log_2\frac{1}{31} = 0.205593 \text{ bits}$$

Ejemplo (2 de 2)

Pero la entropía usando el modelo probabilístico Markoviano es:

$$H(S) = \sum_{i=1}^{N} P_i \cdot H(S_i)$$

En nuestro caso:

$$H(S_B) = -0.01 \cdot \log_2 0.01 - 0.99 \cdot \log_2 0.99 = 0.0807931 \text{ bits}$$
$$H(S_N) = -0.03 \cdot \log_2 0.03 - 0.7 \cdot \log_2 0.7 = 0.881291 \text{bits}$$

$$H = \frac{30}{31} 0.0807931 + \frac{1}{31} 0.881291 = 0.106616 \text{ bits}$$

Es decir, prácticamente la mitad que la que obtendríamos suponiendo independencia.

Distribución de las longitudes de recorridos

La mínima longitud de recorrido de blancos es 1, y ocurre cuando, nada más iniciarse se produce una transición al negro. La probabilidad de una longitud de recorrido de blancos igual a n es:

$$\left(P_{B|B}\right)^{n-1} P_{N|B}$$

La probabilidad de un recorrido menor o igual a n:

$$\sum_{i=1}^{n} \mathrm{P}_{B|B}^{i-1} \cdot P_{N|B} = \frac{\mathrm{P}_{N|B}\left(1-\mathrm{P}_{B|B}^{n}\right)}{1-\mathrm{P}_{B|B}} \qquad \lim_{n\to\infty} \frac{\mathrm{P}_{N|B}\left(1-\mathrm{P}_{B|B}^{n}\right)}{1-\mathrm{P}_{B|B}} = \frac{\mathrm{P}_{N|B}}{1-\mathrm{P}_{B|B}} = 1$$

La longitud media del recorrido es:

$$E[L_B] = \sum_{i=1}^{\infty} i \cdot P_{B|B}^{i-1} \cdot P_{N|B} = \frac{P_{N|B}}{\left(1-P_{B|B}\right)^2} = \frac{1}{P_{N|B}}$$

Ejemplo: facsímil unidimensional

- ★ Cada línea del documento se representa como una secuencia de recorridos alternativos de blanco y negro.
- ★ El primer recorrido es siempre blanco, por lo que si el primer píxel es negro se supone que le precede un recorrido de 0 blancos.
- ★ Las longitudes de cada recorrido tienen diferente probabilidad, por lo que las longitudes se codifican utilizando un código de longitud variable (Huffman).
- ★ Como el número de posibles longitudes es enorme, para evitar un libro de códigos tan grande, cada recorrido r se codifica por un par (m,t) donde $r=m\times64+t$.
- ★ Los m (“make up codes”) sólo se utilizan para recorridos mayores de 64. En otro caso sólo se utiliza t (“terminanting codes”).
- ★ Se utilizan códigos diferentes para los recorridos blancos y negros.
- ★ La norma incluye algunos detalles adicionales que no vamos a detallar aquí.

Codificación

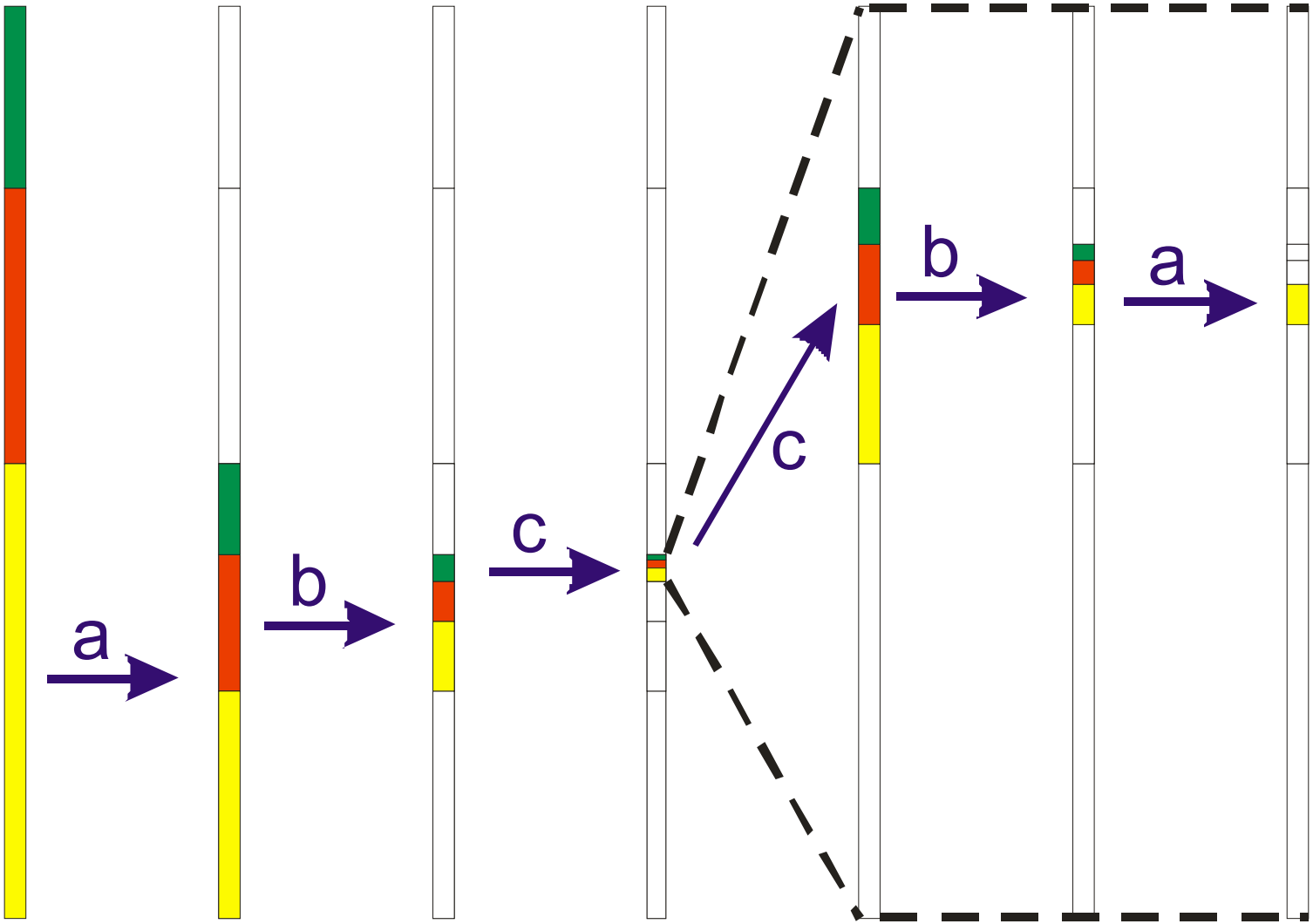
Codificación Aritmética

Codificación aritmética

- Se basa en representar cada posible mensaje como un intervalo de números reales comprendidos entre 0 y 1. La idea básica de la codificación aritmética puede encontrarse en el código de Elías.
- Supongamos un alfabeto con N símbolos, cada uno de ellos con probabilidad p_i.

  Se divide el intervalo unidad en N subintervalos cuya longitud coincide con su probabilidad.
- Cada uno de los subintervalos puede, a su vez, subdividirse en N subintervalos repartiendo de nuevo la longitud proporcionalmente a la probabilidad de cada símbolo y así sucesivamente.
- Se obtiene de esta manera una asociación de cada posible cadena de m símbolos con un subintervalo del intervalo unidad.

Codificación aritmética



Codificación aritmética

A cada cadena fuente, s, se le asocia un intervalo [C(s),C(s)+A(s)).

- C(s) representa el extremo inferior del intervalo
- A(s) la anchura del intervalo

La construcción de los intervalos es recursiva.

Si ξ representa la cadena vacía:

$A(\xi)=1$	$C(\xi)=0$
$A(\alpha_i)=p_i$	$C(\alpha_i)=\sum_{j=0}^{i-1} p_j$
$A(s\alpha_k)=A(s)\times A(\alpha_k)$	$C(s\alpha_k)=C(s)+A(s)\times C(\alpha_k)$

Codificación aritmética

La decodificación puede realizarse siguiendo un procedimiento recurrente similar:

Si m es un número real en el intervalo [0,1), definimos la función S que asocia un símbolo a cada m de la forma:

$$S(m) = \alpha_i \quad \Leftrightarrow \quad C(\alpha_i) \le m \wedge C(\alpha_{i+1}) > m$$

$$D(m) = S(m) D\left(\frac{m - C(S(m))}{A(S(m))} \right)$$

D(m) produce una cadena infinita de símbolos.

El final del proceso de decodificación se alcanza

- bien porque se conoce la longitud de la cadena a decodificar
- bien porque se ha acordado algún otro protocolo entre codificador y decodificador. Por ejemplo (reservando un símbolo para indicar fin de mensaje)

Codificación aritmética: ejemplo

Alfabeto fuente {a,b,c} con probabilidades {0.5,0.3,0.2}.

A(a)=0.5	A(b)=0.3	A(c)=0.2
C(a)=0	C(b)=0.5	C(c)=0.8

Codificación de la cadena “abccba”: [0.3970,0.3979)

α	C(α)	A(α)	C(s)	A(s)
a	0	0.5	0	0.5
b	0.5	0.3	0.25	0.15
c	0.8	0.2	0.37	0.03
c	0.8	0.2	0.394	0.006
b	0.5	0.3	0.397	0.0018
a	0.0	0.5	0.397	0.0009

Codificación aritmética: ejemplo

A partir del número real m=0.3974, conociendo que la cadena la forman 6 símbolos seguiríamos el siguiente procedimiento:

- Por ser *m* menor que 0.5 el primer símbolo del mensaje debe ser "a".
- El siguiente símbolo se determina dividiendo el desplazamiento desde la base del intervalo m-C(a)=0.3974 por la anchura del intervalo (A(a)=0.5). El resultado, 0.7948 está comprendido entre C(b)=0.5 y C(c)=0.8, por lo que el segundo símbolo debe ser "b".
- La diferencia entre la base del intervalo (0.5) y el valor en curso (0.7948) es 0.2948, que, dividido entre la anchura del intervalo, 0.3 es 0.9827, superior a 0.8, luego el tercer símbolo es "c".
- 0.9827-C(c)=0.9827-0.8=0.1827, 0.1827/A(c)=0.1827/0.2=0.91333>0.8=C(c), luego el siguiente símbolo es "c".
- 0.91333-C(c)=0.91333-0.8=0.11333, 0.11333/A(c)=0.56666>0.5=C(b), luego el siguiente símbolo es "b".
- 0.566666-C(b)=0.06666, 0.06666/A(b)=0.22222>0.=C(a), luego el siguiente símbolo es "a".

Codificación aritmética

Para su realización práctica se requiere:

- Transmisión y recepción incremental: el codificador debe poder comenzar la transmisión sin necesidad de examinar toda la cadena a codificar.
- Ser realizable con arquitecturas simples. Para permitir la decodificación unívoca, el intervalo y su anchura deben representarse con tanta más precisión cuanto mayor es la longitud de la cadena.
- Permitir realizaciones eficientes

Para conseguir los objetivos anteriores pueden introducirse ligeras modificaciones al esquema expuesto.

Para apoyar la descripción que sigue codificaremos los primeros símbolos de una fuente binaria que emite "0" con probabilidad 0.6 y 1 con probabilidad 0.4.

Codificación aritmética

S utiliza aritmética entera. El intervalo de partida [low,high) se representa por secuencias binarias. Supongamos inicialmente low=0.000000.... y high=0.111111...

Solamente se mantiene cierto número de bits (por ejemplo 8, 16, o 32) almacenados en los registros. Conforme se codifican símbolos los valores de low y high se acercan, los primeros bits de low y high coinciden, pudiendo transmitirse.

En la tabla que sigue se presenta el proceso (ideal) de codificación de la secuencia "001".En color azul se indican los bits emitidos hacia el decodificador. Los registros low y high mantienen los n bits más significativos que todavía no se han emitido (en el ejemplo n=8).

S.	*low*	*high*	*low(binario)*	*high(binario)*
0	0	0.6	0.00000000000000000000...	0.10011001100110011001...
0	0	0.36	0.00000000000000000000...	0.01011100001010001111...
1	0.216	0.36	0.00110111010010111100...	0.01011100001001000000...

Codificación aritmética

Existe la posibilidad de que los valores correspondientes a high y low requieran una precisión arbitrariamente grande sin que coincidan (y por tanto puedan transmitirse y eliminarse) los primeros bits.

Por ejemplo, si , en el ejemplo anterior los siguientes símbolos a codificar son "0010010001" la tabla se transforma en:

S.	*low*	*high*	*low (binario)*	*high(binario)*
0	0	0.6	0.00000000000000000000	0.10011001100110011001
0	0	0.36	0.00000000000000000000	0.01011100001010001111
1	0.216	0.36	0.00110111010010111100	0.01011100001001000000
0	0.216	0.3024	0.00110111010010111100	0.01001101011010100001
0	0.216	0.26784	0.00110111010010111100	0.01000100100100010010
1	0.247104	0.26784	0.00111111010000100011	0.01000100100100010010
0	0.247104	0.2595456	0.00111111010000100011	0.01000010011100011001
0	0.247104	0.25456896	0.00111111010000100011	0.01000001001010110110
0	0.247104	0.25158298	0.00111111111100100101	0.01000000011001111011
1	0.24979139	0.25158298	0.00111111111100100101	0.01000000011001111011

Codificación aritmética

el límite inferior es de la forma ccc...cccabb...bssssss y el superior ccc...cccbaa...asssssss donde "c" representa un bit comun en los límites superior e inferior, a y b bits complementarios y s bits significativos.

Si no se toma ninguna precaución se produce overflow en el sistema: No puede decidirse que bits han de transmitirse y no se dispone de espacio para almacenarlos.

Para evitar este problema las realizaciones prácticas incluyen alguna forma de evitar este problema:

- introducir un 0 después de una secuencia de N unos seguidos, y transmitirlos.
- incluir un tercer registro en el que se lleva la cuenta de bits de la forma "abbbb" de manera que pueda mantenerse una representación exacta de límites superior e inferior arbitrariamente próximos entre sí sin perder exactitud en la representación.

Comparación cod. aritmética vs. Huffman

Para extensiones de orden m:

Huffman $$H(X) \le l_H \le H(X) + \frac{1}{m}$$

Aritmética $$H(X) \le l_A \le H(X) + \frac{2}{m}$$

Pero Huffman

- Requiere la construcción del código para todas las posibles secuencias, lo que hace inviables extensiones grandes.
- Es más simple de construir
- Es quasi óptimo si las probabilidades se aproximan a potencias de dos
- Puede dar lugar a palabras código de longitud muy grande
- Es más difícil adaptar los códigos Huffman a fuentes cuya estadística varía

Codificación

Códigos no bloque: códigos basados en diccionarios

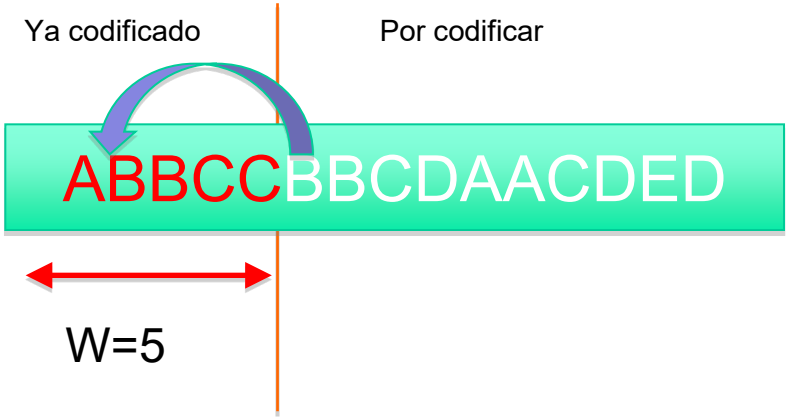
Lempel-Ziv

Son algoritmos adaptativos basados en diccionarios

- Sliding Window Lempel -Ziv (LZ77 o LZ1)
  - El diccionario es parte de la secuencia previamente codificada
- Tree Structured Lempel-Ziv (LZ78 o LZ2)
  - Construye una diccionario a partir de toda la secuencia previamente codificada
- LZW
  - Mejora del LZ78

LZ77

- El diccionario es una porción de la secuencia previamente transmitida (la memoria de búsqueda) que se limita a los últimos W caracteres transmitidos
- El algoritmo analiza los siguientes S caracteres a transmitir, buscando la secuencia más larga posible de caracteres de S que están en W.
- Una vez localizada la secuencia se transmite:
  - Donde comienza dentro de la ventana de búsqueda (4 en el ejemplo)
  - Su longitud (3 en el ejemplo)
  - El primer carácter de S que sigue a la secuencia (D en el ejemplo)



Bibliografía

- S. Khalid, "***Introduction to Data Compression***", 3rd ed., Elsevier, San Francisco, 2006.
- D. Salomon, "***Data Compression. The complete reference***", 2nd ed., Springer, New York, 2000.
- G.A.Jones and J.M. Jones, ***Information and Coding Theory***, Springer Undergraduate Mathematics Series, Springer, 2000.
- T. M. Cover and J.A. Thomas, ***Elements of Information Theory***, 2nd ed., Wiley & Sons, New Jersey, 2006.