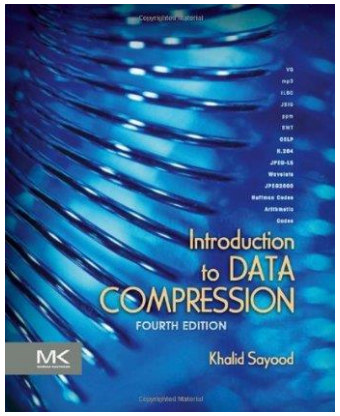
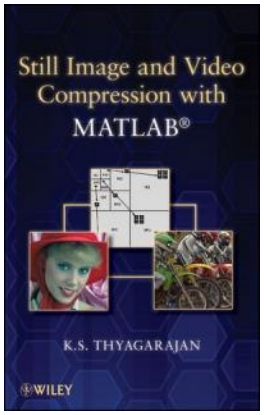
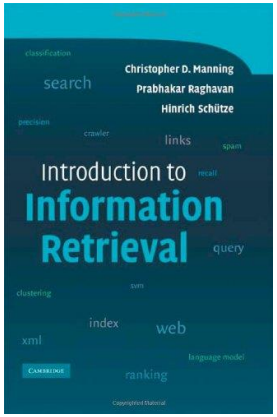
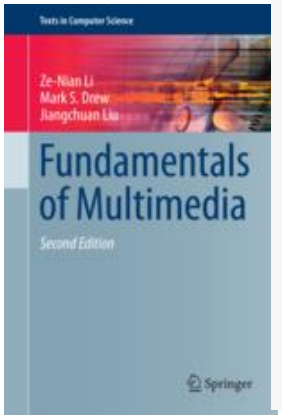









# Tema 4: Codificación Aritmética

Rafael Molina

Depto Ciencias de la Computación

e Inteligencia Artificial

Universidad de Granada

# Contenidos

# I. Objetivos del tema

En este tema vamos a estudiar otros códigos de longitud variable (VLC) , los llamados códigos aritméticos.

Estos códigos son especialmente útiles cuando trabajamos con alfabetos pequeños o con distribuciones de probabilidad muy sesgadas (skew).

Estudiaremos los aspectos básicos de estos códigos, sus propiedades. Compararemos los códigos aritmético y Huffman.

Una variante del código aritmético es el código usado por el estándar Joint Bi-level Experts Group (JBIG) para codificar imágenes binarias, describiremos las partes esenciales de este estándar.

# II. Introducción.

- La codificación Huffman garantiza una longitud media de código R, como mucho superior en 1 bit a la entropía. Otra cota alternativa es la entropía más $p_{max}$+0.086 donde $p_{max}$ es la probabilidad del símbolo que ocurre con mayor probabilidad.

- Usando la segunda cota, si el valor de $p_{max}$ es pequeño, la diferencia entre la codificación Huffman y entropía es pequeña, sin embargo para distribuciones muy descompensadas (sesgadas o skew) $p_{max}$ puede ser bastante grande y el algoritmo de Huffman se hace bastante ineficiente. Observa además que la codificación Huffman asigna como mínimo un bit a cada elemento del alfabeto.

Podemos agrupar los símbolos o letras del alfabeto de forma que la distribución no esté tan descompensada, pero esto no siempre funciona.

Ejemplo:

Consideremos una fuente S con el alfabeto $A=\{a_1,a_2,a_3\}$ con

$$P(a_1)=0.95,\ P(a_2)=0.02 \text{ y } P(a_3)=0.03$$

Se cumple

$$H(S)=0.335 \text{ bits/símbolo.}$$

Una codificación Huffman para esta fuente es

| Letra | Código |
|---|---|
| $a_1$ | 0 |
| $a_2$ | 11 |
| $a_3$ | 10 |

Esta codificación tiene una longitud media de 1.05 bits/símbolo. Este valor es 3.13 veces la entropía

Podríamos agrupar los símbolos de dos en dos y obtener la siguiente tabla de probabilidades.

1.222 bits/símbolo es la longitud media del código, para cada dos símbolos. Es decir, 0.611 bits por símbolo, que es 1.82 veces la entropía

| Letra | Probabilidad | Código |
|---|---|---|
| $a_1a_1$ | 0.9025 | 0 |
| $a_1a_2$ | 0.0190 | 111 |
| $a_1a_3$ | 0.0285 | 100 |
| $a_2a_1$ | 0.0190 | 1101 |
| $a_2a_2$ | 0.0040 | 110011 |
| $a_2a_3$ | 0.0006 | 110001 |
| $a_3a_1$ | 0.0285 | 101 |
| $a_3a_2$ | 0.0006 | 110010 |
| $a_3a_3$ | 0.0009 | 110000 |

La longitud del media del código bajaría a valores próximos a la entropía si usásemos 8 símbolos juntos, pero esto corresponde a 6561 valores a codificar.

Piensa en la complejidad de la codificación y la cabecera del fichero codificado usando Huffman!!

Es más eficiente generar palabras de código por grupos o secuencias de símbolos conforme éstos van apareciendo.

- En la codificación aritmética, que ahora veremos, le asignaremos una etiqueta a la secuencia completa que queremos codificar, esta etiqueta será una fracción de la unidad.

- Generaremos un código para una secuencia de longitud L sin que tengamos que generar código para todas las posibles secuencias de longitud L.

# III. Codificación de una secuencia usando codificación aritmética

- La historia del descubrimiento la codificación aritmética y la resolución del problema de la precisión finita de las representaciones en un ordenador es muy interesante. Ver el libro de Sayood (páginas 93-94).
- Para distinguir una secuencia de símbolos del alfabeto de otra tenemos que etiquetar cada secuencia con un identificador único.
- Un conjunto posible de etiquetas para representar las secuencias de símbolos es el conjunto de los números reales en el intervalo unidad [0,1).
- Asignaremos, por tanto, a cada secuencia de símbolos un número en el intervalo [0,1).

# III.1 Algoritmo de codificación aritmética

Necesitaremos:

- una función que aplique secuencias de símbolos en el intervalo unidad.

Comenzamos asignando un intervalo a cada posible letra del alfabeto. Es decir, si $A=\{a_1,\ldots,a_m\}$ le asignamos a cada símbolo o letra el intervalo

$$a_i \quad i=1,\ldots,m \longrightarrow \left[\sum_{k=0}^{i-1} P(a_k), \sum_{k=0}^{i} P(a_k)\right)$$

donde hemos introducido $P(a_0)=0$. Estos intervalos corresponden obviamente a la función de distribución.

# Con un ejemplo

| Símbolo | Probabilidad | Intervalo en [0,1) | Rango inferior | Rango superior |
|---|---|---|---|---|
| $a_1$ | 0.6 | [0,0.6) | 0.0 | 0.6 |
| $a_2$ | 0.1 | [0.6,0.7) | 0.6 | 0.7 |
| $a_3$ | 0.3 | [0.7,1.0) | 0.7 | 1.0 |

Si queremos codificar la secuencia de un solo símbolo $a_1$ bastaría con que enviásemos un número en [0,0.6).

Si queremos codificar la secuencia de un solo símbolo $a_2$ bastaría con que enviásemos un número en [0.6,0.7).

Si queremos codificar la secuencia de un solo símbolo $a_3$ bastaría con que enviásemos un número en [0.7,1.0).

¿Qué hacemos si queremos codificar dos o más símbolos?

Antes de ver como codificamos una secuencia de símbolos, recordemos que la siguiente transformación lleva el intervalo [0,1) al intervalo [bajo,alto)

$$f : [0,1) \rightarrow [bajo, alto)$$

$$x \rightarrow bajo + (alto - bajo) * x$$

Por ejemplo, si tenemos la siguiente tabla de probabilidades

| Símbolo | Probabilidad | Intervalo en [0,1) | Rango inferior | Rango superior |
|---|---|---|---|---|
| $a_1$ | 0.6 | [0,0.6) | 0.0 | 0.6 |
| $a_2$ | 0.1 | [0.6,0.7) | 0.6 | 0.7 |
| $a_3$ | 0.3 | [0.7,1.0) | 0.7 | 1.0 |

y queremos llevarlas al intervalo [bajo,alto)=[0.0,0.6)

$$f([0.0, 0.6)) = [0.0 + 0.6 * 0.0, 0.0 + 0.6 * 0.6) = [0.0, 0.36)$$

$$f([0.6, 0.7)) = [0.0 + 0.6 * 0.6, 0.0 + 0.6 * 0.7) = [0.36, 0.42)$$

$$f([0.7, 1.0)) = [0.0 + 0.6 * 0.7, 0.0 + 0.6 * 1.0) = [0.42, 0.6)$$

Piensa en la siguiente tabla para entender el siguiente algoritmo

| Símbolo | Probabilidad | Intervalo en [0,1) | Rango inferior | Rango superior |
|---|---|---|---|---|
| $a_1$ | 0.6 | [0,0.6) | 0.0 | 0.6 |
| $a_2$ | 0.1 | [0.6,0.7) | 0.6 | 0.7 |
| $a_3$ | 0.3 | [0.7,1.0) | 0.7 | 1.0 |

# Algoritmo de codificación aritmética

1. bajo=0; alto=1; rango=1;
2. Mientras haya símbolos
    1. Lee símbolo
    2. bajo=bajo+rango*Rango_inferior(símbolo);
    3. alto =bajo+rango*Rango_superior(símbolo)
    4. rango=alto-bajo
3. Envía un código (número) en el intervalo [bajo,alto)

# Ejemplo: Codificación Aritmética

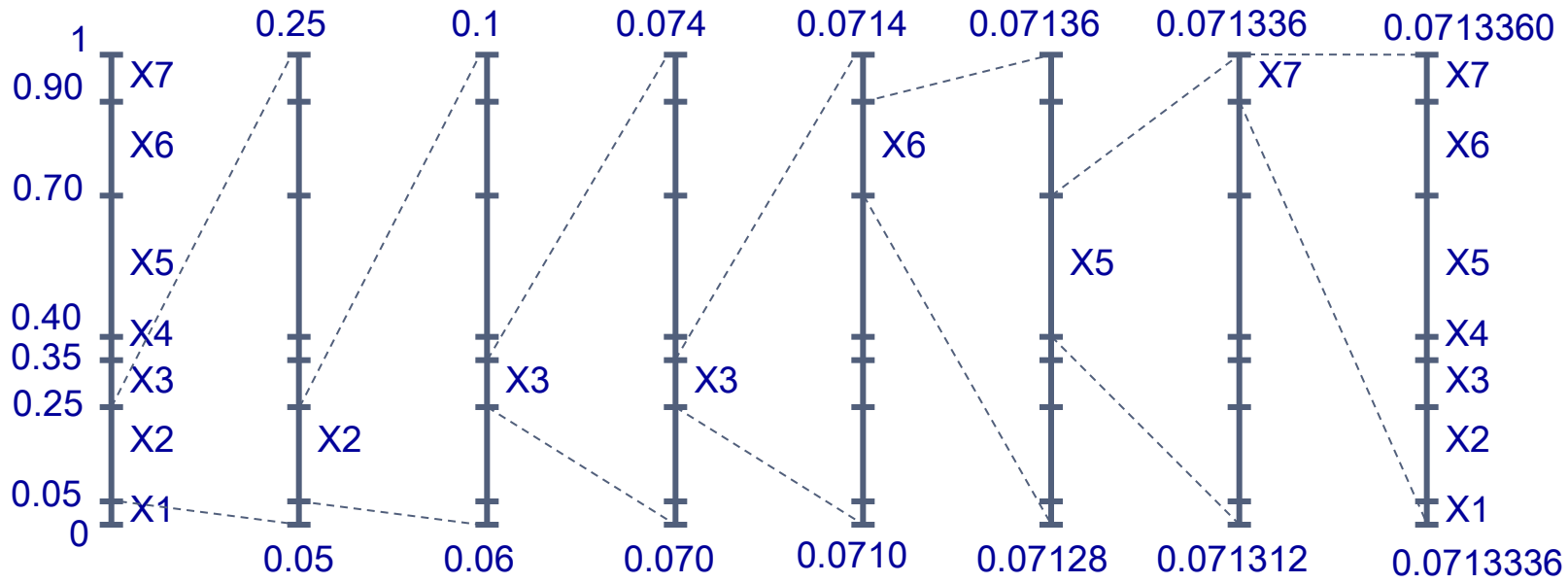
## Codificación Aritmética

| Símbolo | Probabilidad | Intervalo en [0,1) | Rango inferior | Rango superior |
|---|---|---|---|---|
| X1 | 0.05 | [0.0,0.05) | 0.0 | 0.05 |
| X2 | 0.2 | [0.05,0.25) | 0.05 | 0.25 |
| X3 | 0.1 | [0.25,0.35) | 0.25 | 0.35 |
| X4 | 0.05 | [0.35,0.40) | 0.35 | 0.40 |
| X5 | 0.3 | [0.40,0.70) | 0.40 | 0.70 |
| X6 | 0.2 | [0.70,0.90) | 0.70 | 0.90 |
| X7 | 0.1 | [0.90,1.0) | 0.90 | 1.00 |

Secuencia a codificar : *X2 X2 X3 X3 X6 X5 X7*

En este tema estamos usando X1,X2,… y s1,s2,… para notar indistintamente símbolos del alfabeto

# Ejemplo: Codificación Aritmética




Secuencia a codificar: *X2 X2 X3 X3 X6 X5 X7*

Intervalo final = [0.0713336,0.0713360)

# Otro ejemplo

Supongamos que el alfabeto viene dado por $A=\{a_1,a_2,a_3\}$ con probabilidades

$$P(a_1)=0.7,\ P(a_2)=0.1,\ P(a_3)=0.2$$

Los intervalos son entonces [0,0.7), [0.7,0.8), [0.8,1.0)

Supongamos que queremos codificar la secuencia $a_1a_2a_3$

Para codificar $a_1$ sabemos que la etiqueta debe estar en el intervalo

[0,0.7)

EN OTRAS PALABRAS, PASAMOS DEL INTERVALO [0,1) AL [0,0.7)

Recuerda

[0,0.7), [0.7,0.8), [0.8,1.0) y para codificar $a_1$ estábamos en [0,0.7)

Para codificar $a_1a_2$ sabemos que la etiqueta debe estar en

[0+(0.7-0.0)x0.7,0+ (0.7-0.0)*0.8) es decir en [0.49,0.56)

Por último para codificar la secuencia $a_1a_2a_3$ sabemos que la etiqueta debe estar en:

[0.49+(0.56-0.49)x0.8, 0.49+(0.56-0.49)x1) es decir en [0.546, 0.56)

# III.1.1 Generación de etiqueta

- Vamos ahora a realizar el paso 3 del algoritmo de codificación aritmética. Es decir, a asignar un valor en el rango final [bajo,alto).

- Lo haremos utilizando como etiqueta la representación binaria de un número decimal en el rango [0,1)

- Recuerda que la representación binaria de un número decimal en el intervalo [0,1) se obtiene como potencias negativas de 2.

$$.b_1 b_2 b_3 \ldots b_k$$

$$= b_1 2^{-1} + b_2 2^{-2} + b_3 2^{-3} + \ldots + b_k 2^{-k}$$

Veamos un ejemplo.

Ejemplo de representación binaria de números decimales en el intervalo [0,1)

| Número decimal | En binario |
|---|---|
| 0.25 | .0100 |
| 0.625 | .1010 |
| 0.8125 | .1101 |
| 0.9375 | .1111 |

## ALGORITMO DE GENERACIÓN DE ETIQUETA EN [BAJO,ALTO)=[0,1)

etiqueta= representación binaria, valor(etiqueta)= nº decimal

1. etiqueta=0;  k=1;
2. Mientras (valor(etiqueta)<bajo)
    1. Asigna 1 a la posición k-ésima de la etiqueta
    2. Si (valor(etiqueta) > alto)
        1. Reeemplaza el k-ésimo bit de etiqueta por cero
    3. k=k+1

Genera las etiquetas correspondientes a los intervalos [0.30,0.35) y [0,50,0.57)

En nuestro primer ejemplo, el intervalo final es

[0.0713336,0.0713360)

Enviar al decodificador: 0.0713348388671875

16 bits

$$2^{-4}+2^{-7}+2^{-10}+2^{-15}+2^{-16}=$$
$$=0.0001001001000011_2$$

Si comparamos la codificación aritmética con la codificación Huffman de la secuencia:

*X2 X2 X3 X3 X6 X5 X7*

| Símbolo | Probabilidad | Código de Huffman |
|---|---|---|
| X1 | 0.05 | 10101 |
| X2 | 0.2 | 01 |
| X3 | 0.1 | 100 |
| X4 | 0.05 | 10100 |
| X5 | 0.3 | 11 |
| X6 | 0.2 | 00 |
| X7 | 0.1 | 1011 |

Código de Huffman: *01 01 100 100 00 11 1011* 18 bits

Ahora veremos como decodificamos una etiqueta en el intervalo [bajo,alto).

Para a continuación dar respuesta a las siguientes preguntas

- Para una secuencia de letras, ¿qué tamaño tiene su intervalo final [bajo,alto)?

- ¿Cuál es el número mínimo de bits que necesitamos para codificar en binario una etiqueta en el intervalo [bajo,alto).

Observa que es necesario dar respuesta a estas preguntas para poder analizar la longitud de la secuencia codificada.

# III.2 Algoritmo de decodificación aritmética

Recuerda la tabla

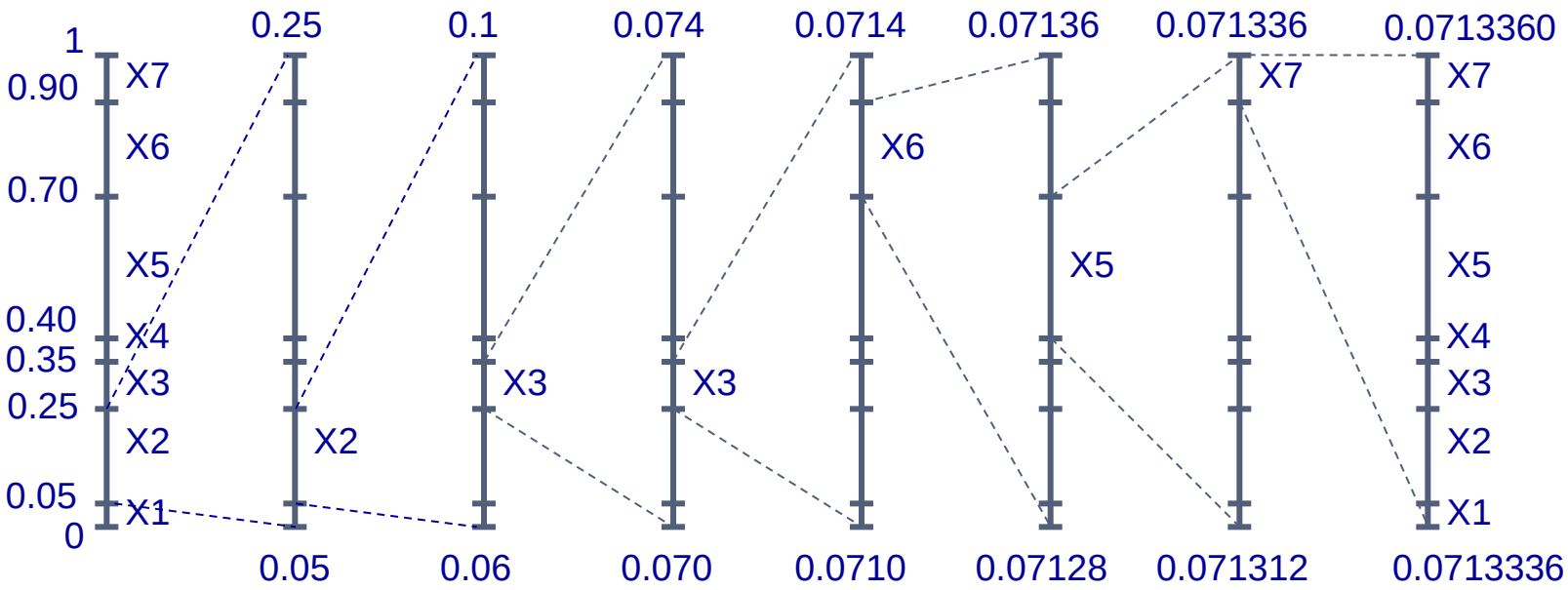
| Símbolo | Probabilidad | Intervalo en [0,1) | Rango inferior | Rango superior |
|---|---|---|---|---|
| $a_1$ | 0.6 | [0,0.6) | 0.0 | 0.6 |
| $a_2$ | 0.1 | [0.6,0.7) | 0.6 | 0.7 |
| $a_3$ | 0.3 | [0.7,1.0) | 0.7 | 1.0 |

## Algoritmo de decodificación aritmetica

1. valor=valor(etiqueta)
2. Repetir
   - Encontrar símbolo s tal que
     rango_inferior(s)<= valor < rango_superior(s)
   - Ouput s
   - bajo=rango_inferior(s)
   - alto=rango_superior(s)
   - rango=alto-bajo
   - valor=[valor-bajo]/rango % lleva valor al intervalo [0,1)
3. Hasta finalizar

Si a<=x<b entonces (x-a)/(b-a) lleva x al intervalo [0,1)

# Ejemplo: Decodificación aritmética



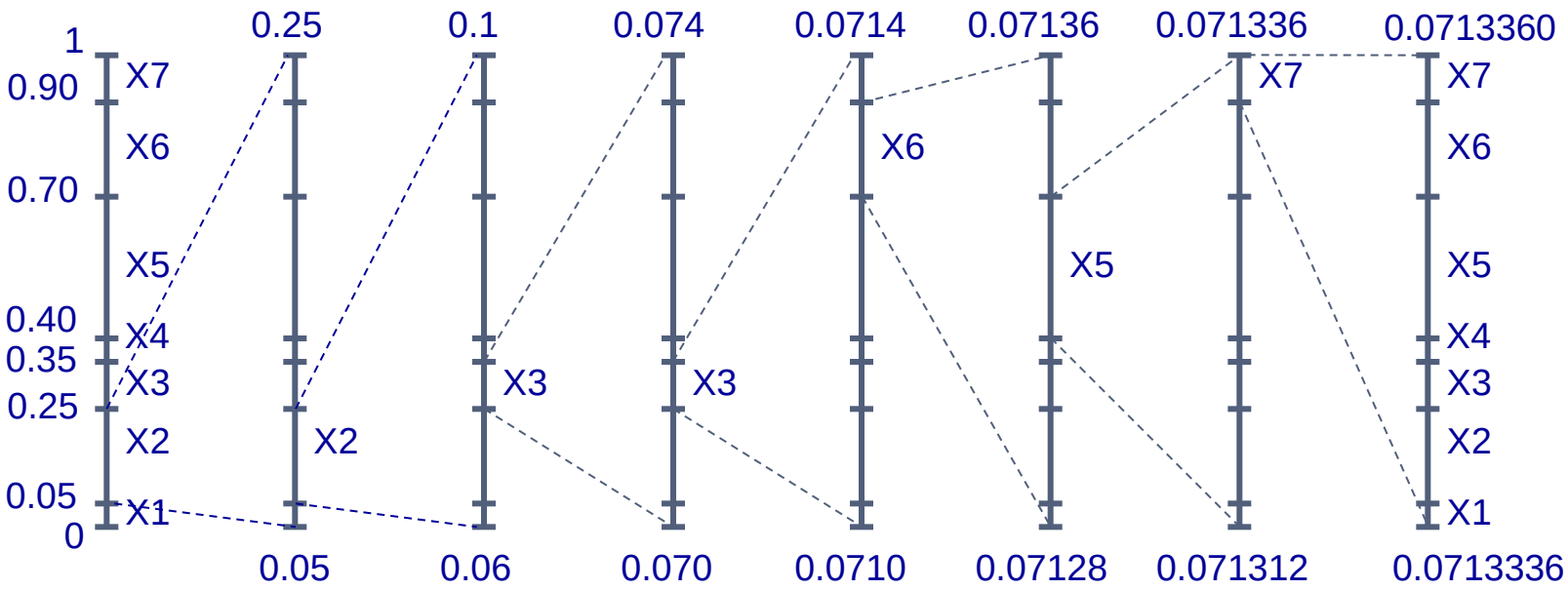

rango_inferior(s)<= valor < rango_superior(s)
Ouput s
bajo=rango_inferior(s)
alto=rango_superior(s)
rango=alto-bajo
valor=[valor-bajo]/rango

valor= 0.0713348388671875

Paso 1

0.05<=valor < 0.25
Output X2
bajo=0.05; alto=0.25
rango=0.20
valor=(valor-bajo)/rango=0.1067

# Ejemplo: Decodificación aritmética

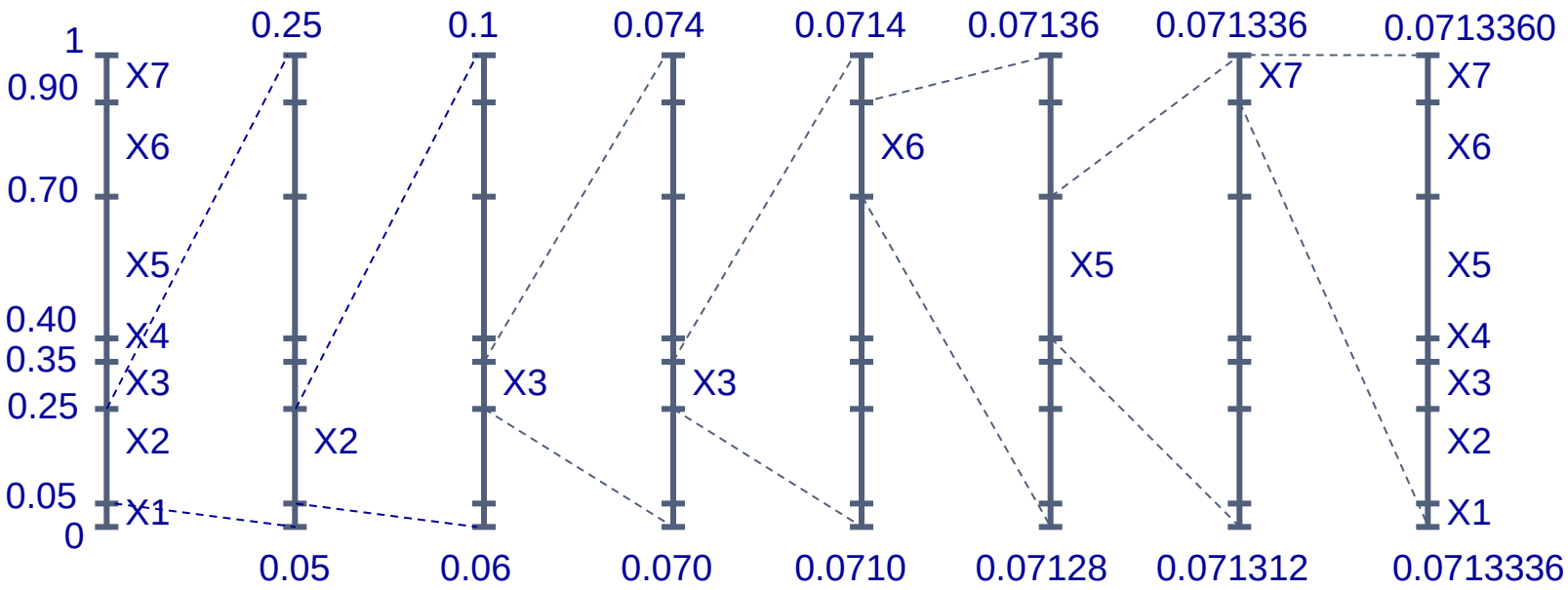



rango_inferior(s)<= valor < rango_superior(s)
Ouput s
bajo=rango_inferior(s)
alto=rango_superior(s)
rango=alto-bajo
valor=[valor-bajo]/rango

Paso 2

0.05<=(valor=0.1067) < 0.25
Output X2
bajo=0.05; alto=0.25
rango=0.20
valor=(valor-bajo)/rango=0.2834

# Ejemplo: Decodificación aritmética




rango_inferior(s)<= valor < rango_superior(s)
Ouput s
bajo=rango_inferior(s)
alto=rango_superior(s)
rango=alto-bajo
valor=[valor-bajo]/rango

Paso 3

0.25<=(valor=0.2834) < 0.35
Output X3
bajo=0.25; alto=0.35
rango=0.10
valor=(valor-bajo)/rango=0.3337

# III.3 Implementación en sistemas de precisión finita

Como hemos visto, el intervalo [bajo,alto) en el que reside la etiqueta de cada secuencia de un tamaño dado es disjunto con los intervalos en los que residirían las etiquetas de cualquier otra secuencia del mismo tamaño.

Cualquier miembro del intervalo [bajo,alto) puede, por tanto, usarse como etiqueta.

¿Qué tamaño tiene el intervalo final [bajo,alto)?

Obviamente el producto de las probabilidades de los símbolos que constituyen la secuencia a codificar

Si la secuencia a codificar es $a_1a_2a_3.....a_M$

$$(alto - bajo) = \prod_{i=1}^{M} P(a_i)$$

¿Cuántos bits necesitaríamos para representar un valor en este intervalo?

# Teorema

Sea [bajo,alto) un intervalo incluido en [0,1) y T=(bajo+alto)/2, observa bajo<T<alto, entonces la representación binaria de T truncada al siguiente número de bits

$$l = \left\lceil \log_2 \frac{1}{(alto - bajo)} \right\rceil + 1$$

que notaremos $\lfloor T \rfloor_l$ cumple

$$bajo \leq \lfloor T \rfloor_l < alto$$

Probemos ahora el resultado enunciado. Probaremos que si

$$bajo \le T < alto \Rightarrow bajo \le \lfloor T \rfloor_l < alto$$

T=(bajo+alto)/2

Recuerda, l el número que se obtiene representando T en binario y truncando a l bits

$$l = \left\lceil \log_2 \frac{1}{(alto - bajo)} \right\rceil + 1$$

Demostración:

como $\lfloor T \rfloor_l \le T$,

la desigualdad de la derecha es cierta.

Para probar la desigualdad de la izquierda haremos dos cosas:

Veremos que para cualquier 0<T<1 y cualquier m entero positivo

1 $$0 \le T - \lfloor T \rfloor_m \le \frac{1}{2^m}$$

Probaremos que

2 $$\frac{1}{2^l} \le \frac{alto - bajo}{2},$$

Lo que nos garantizará que

3 $$bajo \le \lfloor T \rfloor_l$$

1 Probar que para cualquier m entero positivo

$$0 \le T - \lfloor T \rfloor_m \le \frac{1}{2^m}$$

es muy fácil. Además

2
$$\frac{(alto-bajo)}{2} = \frac{1}{\frac{2}{(alto-bajo)}} = \frac{1}{2^{\log\frac{1}{(alto-bajo)}+1}} \ge \frac{1}{2^{\left\lceil \log\frac{1}{(alto-bajo)} \right\rceil+1}} = \frac{1}{2^l}$$

Por tanto $l = \left\lceil \log_2 \frac{1}{(alto-bajo)} \right\rceil + 1$ cumple $\frac{1}{2^l} \le \frac{alto-bajo}{2}$,

Combinando 1 y 2 tendremos

$$0 \le \frac{alto+bajo}{2} - \left\lfloor \frac{alto+bajo}{2} \right\rfloor_l \overset{1}{\underset{2}{\le}} \frac{1}{2^l} \le \frac{alto-bajo}{2} \Rightarrow bajo \le \left\lfloor \frac{alto+bajo}{2} \right\rfloor_l$$

- ¿Tenemos que haber calculado el intervalo [bajo,alto) final para mandar la etiqueta?

Obviamente no, cuando los bits más significativos de bajo y alto coinciden podemos mandarlos y olvidarnos de ellos.

Con un ejemplo si bajo=0.00  alto=0.01, decir, el intervalo [0,1/4) de este intervalo no saldrá nunca y el primer cero no cambiará nunca.

Igualmente si bajo=0.10 y alto=0.11 es decir [0.5, 0.75) de este intervalo no saldrá nunca. Podríamos mandar el primer 1.

Hasta que los dígitos más significativos coincidan no podemos enviarlos. Por ejemplo, [0.00) y [0.11), hay que esperar

Las ideas que acabamos de discutir sobre implementación binaria (o flotante) pueden aplicarse a aritmética entera.

- En lugar del intervalo [0,1) se trabaja en el intervalo [0,N) usando aritmética entera.

- La idea subyacente es evitar operaciones en flotante.

Puedes encontrar más detalles en el libro de Sayood.

# IV. Comparación del código de Huffman y el código aritmético

Nos preguntamos ahora ¿cómo de eficiente es el código que hemos generado?.

Supongamos que codificamos secuencias de longitud m, **x** es la secuencia a codificar y P(**x**), que es el producto de las probabilidades de los símbolos en la secuencia, es (alto-bajo). La longitud media del código aritmético sería

$$l_{A^m} = \sum l(\mathbf{x})P(\mathbf{x}) = \sum \left( \left\lceil \log \frac{1}{P(\mathbf{x})} \right\rceil + 1 \right) P(\mathbf{x})$$

$$\leq \sum \left( \log \frac{1}{P(\mathbf{x})} + 2 \right) P(\mathbf{x}) = -\sum P(\mathbf{x}) \log P(\mathbf{x}) + 2 = H(X^m) + 2$$

y por tanto

$$H(X^m) \leq l_{A^m} \leq H(X^m) + 2$$

de donde, para cada símbolo tenemos

$$H(X) \leq l_A \leq H(X) + 2/m$$

Así pues, aumentando la longitud de la secuencia nos garantizamos una codificación tan próxima a la entropía como queramos.

Si codificamos con bloques de m términos. Sabemos que para la codificación aritmética se cumple

$$H(X) \leq l_A \leq H(X) + 2/m$$

y para el código de Huffman tenemos

$$H(X) \leq l_H \leq H(X) + 1/m$$

En principio parece mejor la codificación Huffman. Sin embargo, recuerda que para el código de Huffman hemos de construir el árbol para todos los posibles $k^m$ valores (supuesto k el número de términos del alfabeto), algo poco práctico.

Recuerda que el código de Huffman será imbatible si las probabilidades son potencias de ½.

La ganancia depende también de la fuente. Sabemos que otra cota superior para el código de Huffman es $0.086+p_{max}$, donde $p_{max}$ es la probabilidad máxima del alfabeto.

Si el alfabeto es grande y las probabilidades no están muy descompensadas la codificación aritmética no es muy ventajosa sobre la codificación Huffman. Pensemos, sin embargo en la codificación de un fax.

Una gran ventaja del código aritmético es la posibilidad de implementar simultáneamente varios códigos aritméticos (sólo necesitamos tener varias tablas de probabilidades).

Además, en la codificación aritmética podemos cambiar más fácilmente las probabilidades conforme se va modificando la entrada.

# V. Codificación Aritmética Binaria

La codificación aritmética binaria aborda la codificación de fuentes binarias usando codificación aritmética.

Observa que al ser la fuente binaria sólo necesitamos conocer la probabilidad de un símbolo.

Para la codificación aritmética binaria eficiente se han propuesto diferentes métodos:

- Los algoritmos Q, QM y MQ se desarrollaron para JBIG, JBIG2 y JPEG-2000.
- El algoritmo M, más conocido por CABAC (context-based adaptive binary arithmetic coder), se utiliza en H.264 y H.265.

# VI. Codificación Aritmética Adaptativa

Observa que nada nos impide que, codificado un símbolo, utilicemos para el siguiente símbolo a codificar una tabla de probabilidades que dependa del símbolo anterior codificado.

Esta idea conduce al código aritmético adaptativo que simplemente se basa en utilizar, dado x(n), la tabla de probabilidades Pr(x|x(n)) para codificar el próximo símbolo.

Observa que en este caso codificador y decodificador necesitan conocer todas las tablas de probabilidad utilizadas.

# VII. Aplicaciones

La codificación aritmética se utiliza en diferentes aplicaciones de compresión sin y con pérdida. Es parte de diferentes estándares.

Existen diferentes organismos que proporcionan estándares:
International Standards Organization (ISO)
International Electrotechnical Comission (IEC)
Son grupos industriales que trabajan en estándares multimedia.

International Telecommunications Union (ITU) es parte de las Naciones Unidas y trabaja en estándares multimedia para los miembros de las Naciones Unidas.

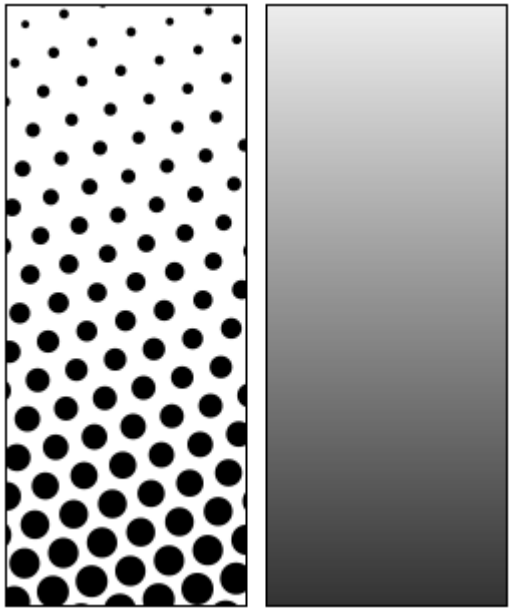
Veamos ahora una aplicación concreta. El standard JBIG.

Como curiosidad, había una implementación de JPEG que usaba codificación aritmética pero no se utilizó por temas de patentes.
Visita http://en.wikipedia.org/wiki/Arithmetic_coding

# VII.I El standard JBIG

- Joint Bi-level Image expert Group.
- Codificación para imágenes binarias.
- Gráficos e imágenes usando medios tonos (half-tones).


**Halftone** es una técnica que simula tonos continuos mediante el uso de puntos de diferentes tamaños, forma o espaciado



Izquierda, puntos halftone, Derecha, como lo ve el ojo humano a una cierta distancia

# Características y elementos de JBIG

## Características de JBIG:

- ITU-T Recomendación T.82 (1993)
- ISO/IEC Estándar internacional 11544
- Pensado para fax, transmisión y almacenamiento
- Afectado por 24 patentes, principalmente de IBM y AT&T

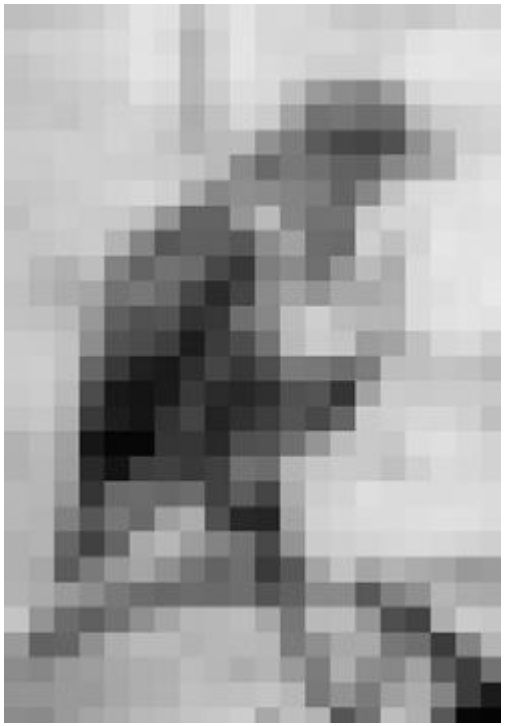
## Elementos de JBIG:

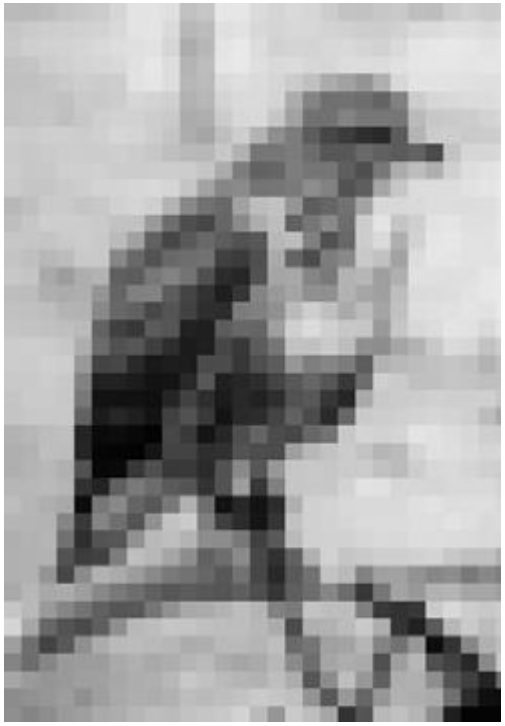
- Codificación progresiva (multiresolución)
- Codificación aritmética adaptativa (algoritmo QM)
- Probabilidades seleccionadas mediante patrones

# ¿Qué es la decodificación progresiva?

Decodificación secuencial

# ¿Qué es la decodificación progresiva?

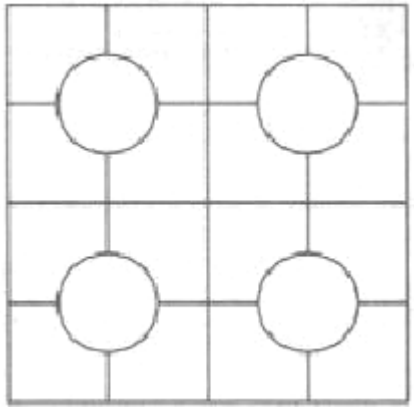




Decodificación progresiva

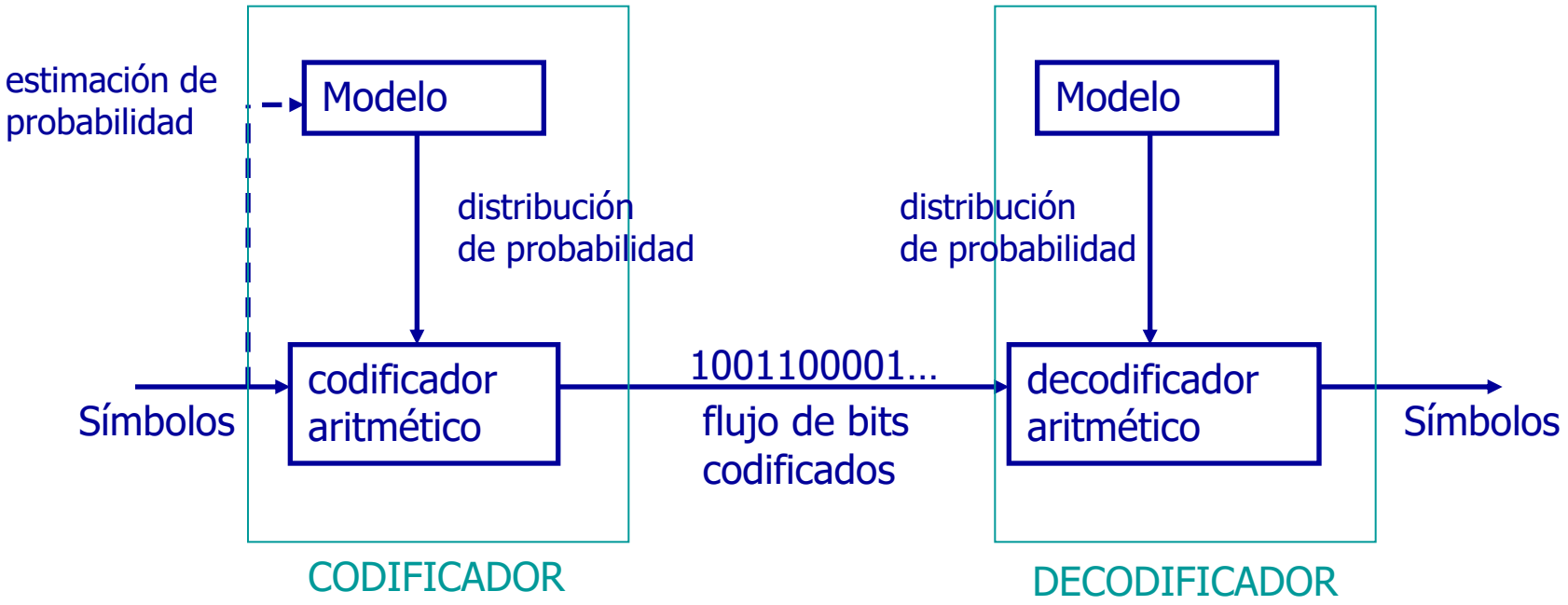
# Reducción de resolución o construcción de capas

Agrupar los píxeles en bloques de 2x2



A los píxeles de baja resolución, los círculos, se les asigna un valor (0 o 1) en función de los valores en la capa de más resolución  y los valores de baja resolución ya codificados

# Diagrama de la Codificación JBIG



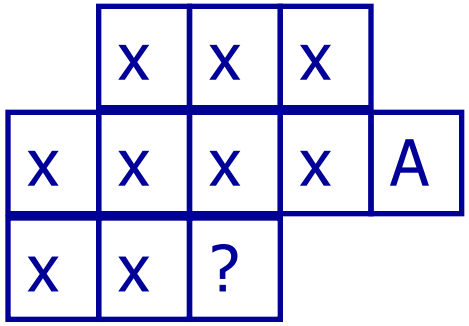

La distribución de probabilidad de los píxeles blancos y negros se determinan de forma adaptativa basándose en subconjuntos de píxeles que ya han sido codificados

# Características de JBIG

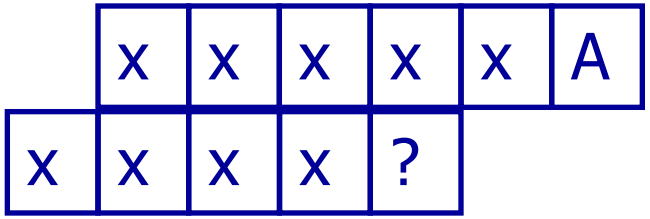
- En codificación aritmética es muy fácil separar el *modelo* (probabilidades) y el *codificador/decodificador*.
- JBIG usa una modificación del codificador aritmético (llamado codificador QM) con múltiples modelos (1024 o 4096 modelos, dependiendo de la resolución de la imagen).
- La distribución de probabilidad de los píxeles blancos y negros se determinan de forma adaptativa basándose en píxeles del vecindario que ya han sido codificados.

# Patrones de contexto para la capa base (la de menor resolución)

Patrón de 10 píxeles




Patrón de 10 píxeles en dos líneas

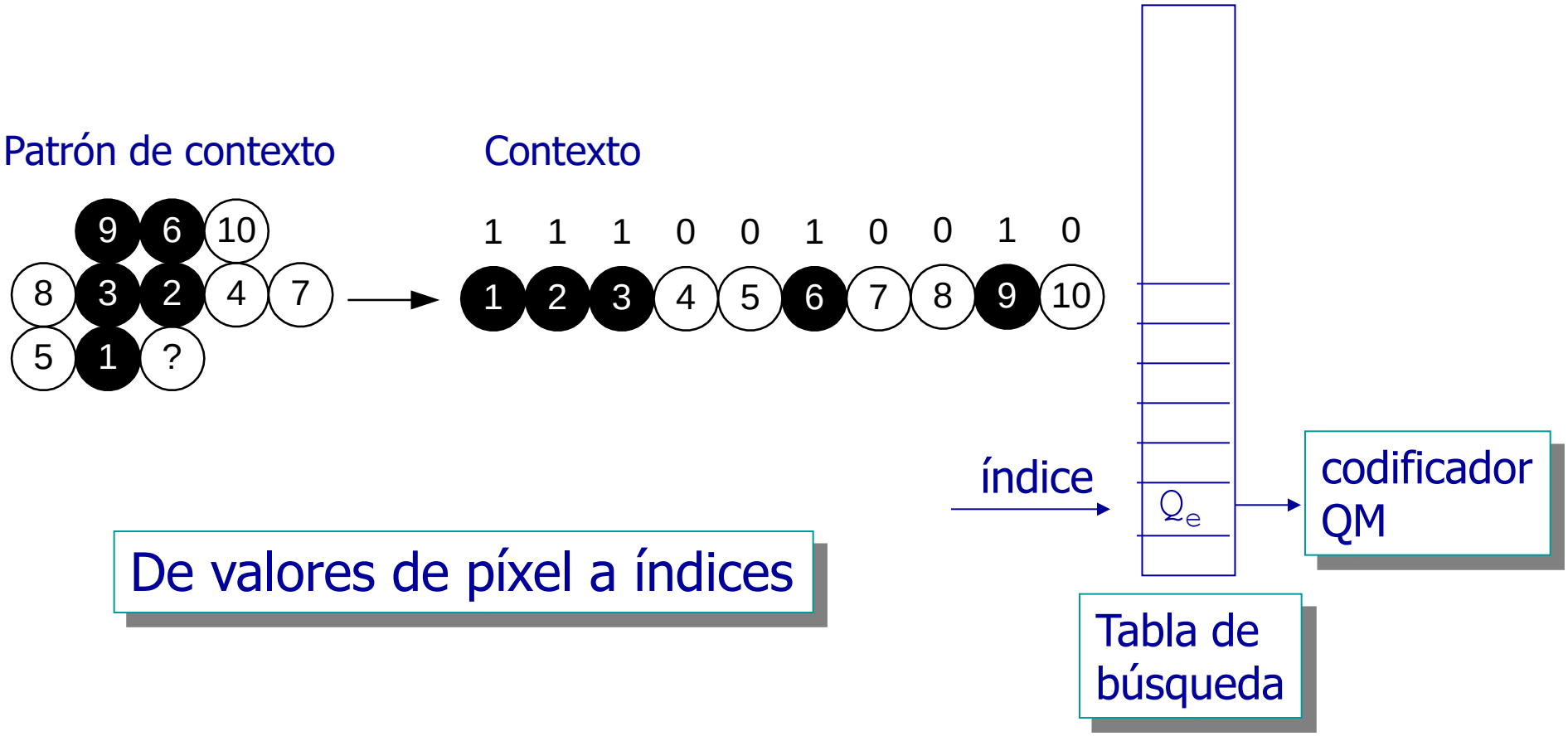



? Píxel a codificar

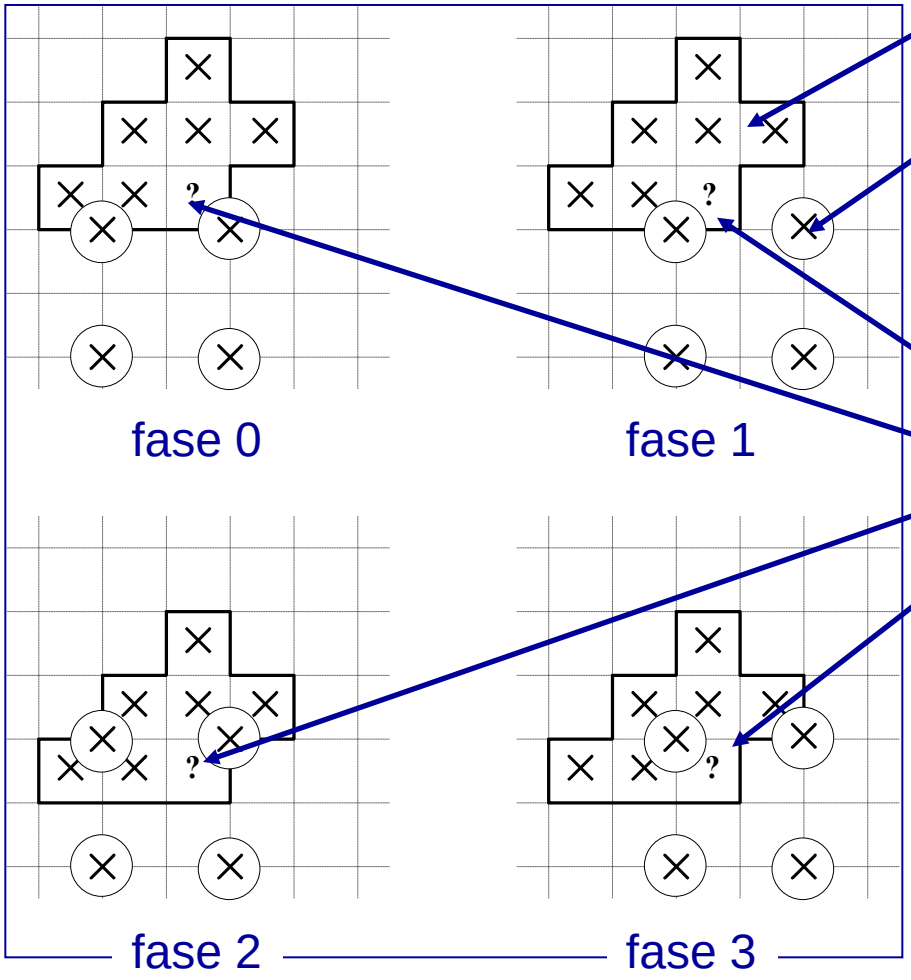
x Píxel ya codificado en la posición relativa a ? marcada

A Píxel adaptativo (AT): un píxel flotante en un vecindario (ya codificado) del patrón. Es útil para captar estructuras periódicas de la imagen.

# Patrones de contexto para la capa base



# Patrones de contexto para las capas diferenciales




Pixel ya codificado

Pixel de la capa inferior

Pixels a codificar en esta capa

Usaremos 12 bits
para cada contexto
(10 bits para pixels
+ 2 bits para fase)

# Codificación Imágenes no binarias

Codificación por planos de bits.

Más eficiente si se usan códigos Gray.

Un código Gray asigna a dos números consecutivos representaciones binarias que difieren sólo en un bit.

| Binario | 000 | 001 | 010 | 011 | 100 | 101 | 110 | 111 |
|---|---|---|---|---|---|---|---|---|
| Gray | 000 | 001 | 011 | 010 | 110 | 111 | 101 | 100 |

Para convertir en número binario estándar $b_0b_1b_2...b_n$ a su código Gray $g_0g_1g_2...g_n$ se siguen las ecuaciones:

$g_0 = b_0$

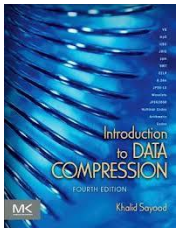
$g_k = b_k$ XOR $b_{k-1}$

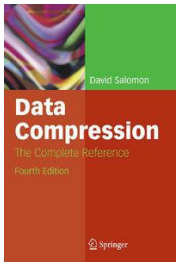
# IX. Bibliografía

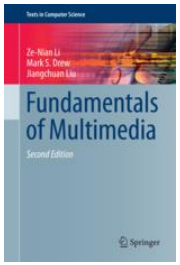

K. Sayood, "Introduction to Data Compression", Morgan and Kaufmann, 2012.

D. Salomon, "Data Compression: The Complete Reference, Springer, 2006

Ze-Nian Li, M. S. Drew, J. Liu, " Fundamentals of Multimedia", Springer, 2014



Apuntes de Prof. Javier Mateos, Laboratorio Multimedia, Escuela Técnica Superior de Ingenierías Informática y Telecomunicación, Universidad de Granada

Apuntes de Prof. Luis Torres, Codificación de Contenidos Audiovisuales, Escuela Técnica Superior de Ingeniería de Telecomunicación de Barcelona, Universidad Politécnica de Catalunya.