

Almacenamiento de imágenes digitales



Primera parte

Representación de Imágenes Digitales

Representar una imagen de $n \times n$ píxeles mediante una matriz requiere:

- n^2 bits si es binaria
- $n^2 \log_2 (L)$ bits si la imagen es en niveles de gris, con L niveles
- $3n^2 \log_2 (L)$ bits si la imagen es a color (RGB) con L niveles.

En este capítulo estudiaremos representaciones de imágenes que pueden ser más económicas que la representación matricial en determinados casos, ya que se elimina información redundante.

Redundancia de datos

Sean n_P y n_Q el número de bits necesario para almacenar dos representaciones, P y Q, distintas de la misma imagen.

La redundancia relativa de P (respecto de Q) es:

$$R = 1 - (1/C)$$

donde $C = n_P/n_Q$ es el radio de compresión.

Por ejemplo, si $C = 10$, significa que la representación P necesita 10 bits por cada bit de la representación Q. Así, $R = 0,9$ y por tanto, el 90% de los datos en la representación P es redundante.

Observemos que si $n_Q = n_P$, $C = 1$ y $R = 0$, por lo que P no es redundante

Si $n_Q \ll n_P$, entonces $C \rightarrow +\infty$, $R \rightarrow 1$

Si $n_Q \gg n_P$, entonces $C \rightarrow 0$, $R \rightarrow -\infty$

En una imagen digital hay tres tipos de redundancia:

- *Redundancia en el código.*

Código es un sistema de símbolos usado para representar la información. A veces, la longitud de las palabras usadas en el código es mayor de lo necesario.

- *Redundancia entre píxeles.*

Debida a la correlación espacial entre un píxel y sus vecinos.

- *Redundancia psicovisual.*

Parte de la información es ignorada por el ojo humano.



FIGURE 8.1 Computer generated $256 \times 256 \times 8$ bit images with (a) coding redundancy, (b) spatial redundancy, and (c) irrelevant information. (Each was designed to demonstrate one principal redundancy but may exhibit others as well.)

La compresión de imágenes consiste en eliminar una o más de estas redundancias.

Se pueden clasificar las distintas formas de compresión de imágenes en:

- aquellas que no producen errores (o pérdida de información) en la imagen;
- aquellas que sí lo producen.

Los procesos de compresión que eliminan la redundancia de código y/o entre píxeles no producen errores.

Aquellos que eliminan la redundancia psicovisual, sí.

- *Redundancia en el código.*

Código de Huffman

Codificación **sin pérdida** en imágenes en escala de **grises**

- *Redundancia entre píxeles.*

LZW

RLE



Codificación **sin pérdida** en imágenes en escala de **grises**

Representación por bloques.

Representación en árbol cuaternario.

Representación por borde de la imagen.

Conjuntos derivados



Codificación **sin pérdida**
en imágenes **binarias**

Planos de bits

Codificación **sin pérdida** en imágenes en escala de **grises**

- *Redundancia psicovisual.*

Transformada de Fourier

Transformada de Hadamard

Transformada de Walsh-Hadamard

Transformada discreta coseno



Codificación **con pérdida** en imágenes
en escala de **grises**

Compresión de imágenes eliminando la redundancia de código

Sea P una imagen en escala de grises (con L niveles de grises: $0, 1, 2, \dots, L-1$) de N píxeles.

Sea N_k la cantidad de píxeles que tienen el mismo nivel de gris k y $l(k)$ el número de bits necesarios para almacenar el color k . Entonces, el promedio de bits necesarios para representar cada píxel es:

$$\frac{\sum_{k=0}^{L-1} l(k) N_k}{N}$$

Es decir, si llamamos $p(k)$ a la probabilidad de cada píxel de tener el color k , entonces $p(k) = N_k / N$ y la fórmula anterior se puede expresar

$$\sum_{k=0}^{L-1} l(k) p(k)$$

Ejemplo: si $L=8$ entonces el promedio de bits necesario para representar cada píxel es 3, usando un código binario de longitud constante

IDEA: Código de longitud variable tal que a aquellos valores con más probabilidad se le asigna un menor número de bits, entonces se consigue que el promedio sea menor.

Ejemplo: **Código de Huffman.**

Paso 1. Ordenar los valores de grises según la probabilidad de que ocurran (de mayor a menor).

Paso 2. Crear una tabla donde se van sumando sucesivamente las dos probabilidades más pequeñas, hasta que no se pueda más.

Paso 3. Crear un árbol binario a partir de la tabla, donde los hijos son las probabilidades de partida.

Paso 4. A partir del árbol, crear el nuevo código.

Código de Huffman

Ejemplo: Consideremos una imagen con 6 niveles de grises:

$p(a_1)=0,1$	$p(a_2)=0,4$	$p(a_3)=0,06$	$p(a_4)=0,1$	$p(a_5)=0,04$	$p(a_6)=0,3$
--------------	--------------	---------------	--------------	---------------	--------------

Observemos que si usamos el código natural, necesitamos 3 bits para codificar cada valor:

Valor	Valor codificado
a_1	000
a_2	001
a_3	010
a_4	011
a_5	100
a_6	101

Vemos que el promedio de bits necesario es 3.

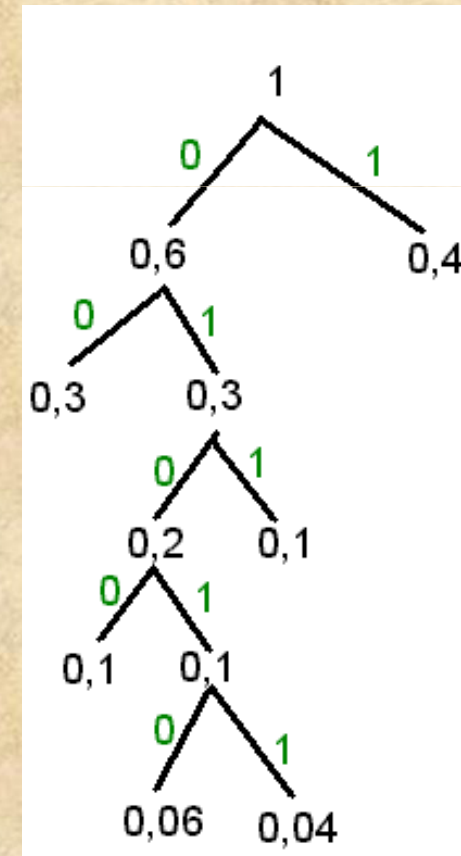
Código de Huffman

Ahora creamos la tabla para obtener el código de Huffman:

Original source		Source reduction				
Symbol	Probability	1	2	3	4	
a_2	0.4	0.4	0.4	0.4	0.6 0.4	
a_6	0.3	0.3	0.3	0.3		
a_1	0.1	0.1	0.2	0.3		
a_4	0.1	0.1				0.1
a_3	0.06	0.1	0.1			
a_5	0.04					

Formamos el árbol binario a partir de la tabla:

Original source			Source reduction							
Symbol	Probability	Code	1		2		3		4	
a_2	0.4	1	0.4	1	0.4	1	0.4	1	0.6	0
a_6	0.3	00	0.3	00	0.3	00	0.3	00	0.4	1
a_1	0.1	011	0.1	011	0.2	010	0.3	01		
a_4	0.1	0100	0.1	0100	0.1	011				
a_3	0.06	01010	0.1	0101						
a_5	0.04	01011								



Código de Huffman

Creamos el nuevo código:

Valor	Probabilidad	Valor codificado con el código de Huffman
a_1	0,1	011
a_2	0,4	1
a_3	0,06	01010
a_4	0,1	0100
a_5	0,04	01011
a_6	0,3	00

Ahora, el promedio de bits necesario es:

$$3(0,1)+0,4+5(0,06)+4(0,1)+5(0,04)+2(0,3)=2,2$$

Luego el radio de compresión sería: $C = 3/2,2 = 1,36$ y la redundancia relativa es: $R = 1 - 1/1,36 = 0,26$. Por tanto, el 26% del primer código era redundante.

Código de Huffman

- La codificación de Huffman consigue el número más pequeño posible de símbolos de código.
- Además cualquier cadena del código es decodificable de manera única.

Por ejemplo, para el código de la tabla anterior, la cadena

010100111100

sólo se puede decodificar por (si se lee de izquierda a derecha):

$a_3 a_1 a_2 a_2 a_6$.

Ejercicio: ¿Cuántos códigos de Huffman distintos hay para un código inicial de tres símbolos? Constrúyelos.

Ejercicio



Esta imagen de 8 bits posee sólo 4 niveles de gris distintos con probabilidades $p(r_1)=0,25$; $p(r_2)=0,47$; $p(r_3)=0,25$; $p(r_4)=0,03$. ¿Se podría dar una representación más económica con un código de longitud constante? ¿Se podría mejorar con un código de longitud variable? ¿Cuáles serían, en cada caso, los radios de compresión y la cantidad de datos redundantes?

Compresión de imágenes eliminando la redundancia entre píxeles

- LZW
- RLE
- Distintas representaciones para imágenes binarias:
 - Representación por filas (RLE).
 - Representación por bloques.
 - Representación mediante el borde de la imagen.
 - Conjuntos derivados.
 - Representación aproximadas.

Codificación LZW (Lempel, Ziv, Welch)

Claves:

La compresión se produce por darse reiteradamente la sustitución de cadenas de caracteres consecutivos en la imagen, por un solo símbolo de código.

Conforme se realiza la codificación, se va creando un diccionario en el que se recogen los nuevos símbolos de código.

En general, reduce la redundancia entre píxeles, excepto si el diccionario es excesivamente largo, en cuyo caso puede resultar contraproducente.

Codificación LZW (Lempel, Ziv, Welch)

Dada una "frase" de un alfabeto de M letras, el pseudocódigo del algoritmo es:

1. Inicializar una tabla (diccionario), asignando a cada letra un código de 0 a M-1.
2. Inicializar P = primera letra de la frase.
3. Sea S el siguiente carácter en la frase
4. Si PS es una palabra del diccionario
 P = PS e ir al paso 3.
 En otro caso,
 añadir PS al diccionario asignándole un código n no utilizado,
 c(PS)=n.
 P=S e ir al paso 3.

Codificación LZW

EJEMPLO: Dado un alfabeto con 3 letras A, B, C. Queremos codificar la palabra ABACABA.

Primero codificamos el alfabeto:
 $c(A)=0$, $c(B)=1$, $c(C)=2$.

Los pasos del algoritmo son



La frase codificada sería
ABACABA=010230
(fijándonos en la primera columna: P).

P	S	Diccionario	Código
		A	0
		B	1
		C	2
A	B	AB	3
B	A	BA	4
A	C	AC	5
C	A	CA	6
A	B	AB	
AB	A	ABA	7
A			

Representación por filas o Run-Length Coding

- Se trata de una técnica desarrollada sobre 1950 y se conoce como run-length encoding, o RLE.
- Este tipo de representación o compresión es apropiado para imágenes en las que aparecen intensidades repetidas a lo largo de las filas de la imagen.
- Así, cada secuencia de la misma intensidad se codifica por un par que especifica las **longitud y el nivel de gris** que se repite en la secuencia.
- En los casos en los que hay pocas repeticiones, la técnica en realidad resulta en un aumento de datos.

Representación por filas o Run-Length Coding

- Supongamos que la fila tiene una longitud n , y que hay r secuencias. Puesto que son necesarios $\log_2 n$ bits para especificar la longitud de una secuencia (puede tener cualquier longitud entre 1 y n), el número de bits necesarios para especificar todas las longitudes de secuencias de la fila es $r \log_2 n$.

- Así si hay L posibles niveles de gris, cada fila requiere

$$r (\log_2 n + \log_2 L) = r \log_2 (n L) \quad \text{bits,}$$

en contraposición con los $n \log_2 L$ bits que son necesarios cuando la fila es tratada como una cadena de longitud n .

- Por tanto, si hay sólo unas pocas secuencias (r es "pequeño"), esta representación es muy económica.

Esta representación resulta especialmente útil en el caso de imágenes binarias.

Representación por
filas. Caso $L=2$.

Sólo hay que indicar el
primer elemento de la fila
y las longitudes de las
secuencias alternadas:
 $1 + r \log_2 n$ bits / fila

- Ejemplo de matriz
simple

```
101111111111100
000011101110000
000000111000000
000011101110000
0011100000011100
```

- Ejemplo de
correspondencia *runs*

```
1 : 1 1 11 2
0 : 4 3 1 3 4
0 : 6 3 6
0 : 4 3 1 3 4
0 : 2 3 5 3 2
```

- Ejemplo de
correspondencia
ArrayRuns

```
5 15 1 1 1 11 2 0
0 4 3 1 3 4 0 0 6
3 6 0 0 4 3 1 3 4
0 0 2 3 5 3 2 0
```

Representación por filas: **Árboles binarios**

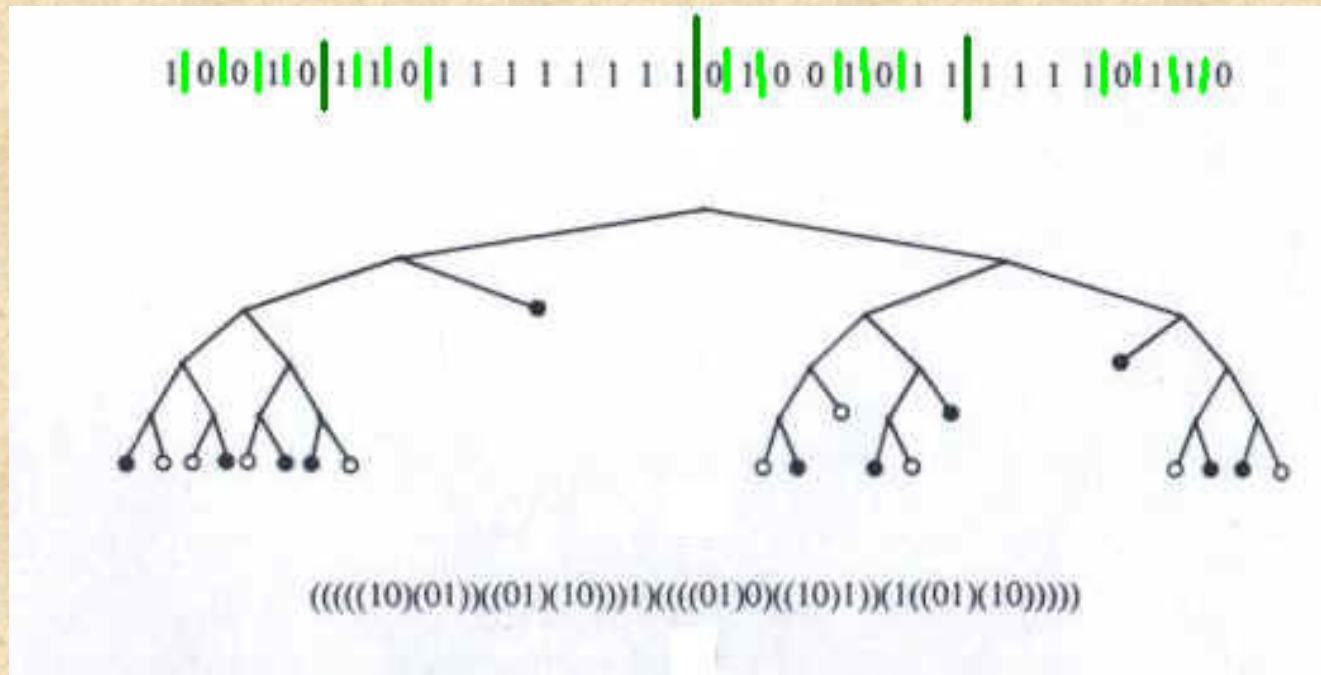
Supongamos, para simplificar, que la longitud de la fila es una potencia de 2, digamos $n = 2^k$.

Método (árbol binario):

- El nodo raíz del árbol representa la fila entera.
- Si toda la fila tiene un valor, etiquetamos el nodo raíz con ese valor y paramos; en este caso, el árbol consiste sólo en el nodo raíz. En otro caso, añadimos dos descendientes al nodo raíz, representando las dos mitades de la fila.
- El proceso se repite entonces para cada uno de esos nuevos nodos: si su mitad de la fila tiene valor constante, lo marcamos con ese valor y no le damos ningún descendiente; si no, le damos dos descendientes correspondientes a las dos mitades de su mitad. En el nivel k , los nodos (si hay alguno) corresponden a píxeles únicos, y son todos nodos hoja, marcados con los valores de sus píxeles.
En general, nodos hoja en el nivel h , significan secuencias de 2^{k-h} *símbolos iguales consecutivos*.

Representación por filas: **Árboles binarios**

EJEMPLO



Representación binaria de una cadena de longitud 32. Los círculos rellenos y huecos del árbol son nodos hoja correspondientes a bloques de 1's y 0's respectivamente.

Recorriendo los nodos hoja de izquierda a derecha, se pueden leer las secuencias de 0's y 1's según el nivel en que se encuentre cada uno.

1:1,2,1,1,2,1,8,1,1,2,1,1,6,1,2,1

El espacio necesario para almacenar el árbol es proporcional al número de nodos. Si la fila consta sólo de unas pocas secuencias, el árbol tendrá relativamente pocos nodos, pero el número exacto depende de las posiciones y longitudes de las secuencias.

Representación por bloques

- **MATs** (*medial axis transformation*)

Sea S una imagen de L niveles de gris.

- A cada punto P de la imagen, le asociaremos un conjunto de cuadrados(u otro polígono regular) de lados de longitud impar n centrados en P .
- Sea S_P el más grande de tales cuadrados que tiene un valor **constante de nivel de gris**, y sea r_P el radio de S_P . Llamaremos a S_P un *bloque maximal*, si no existe otro bloque S_Q de la imagen que lo contenga.
- Si especificamos el conjunto de centros P , radios r_P , y valores v_P de los bloques maximales, la imagen está completamente determinada, puesto que cualquier punto de S se encuentra en al menos un bloque maximal.
- Sólo hacemos esto para $L-1$ de los valores de gris.

Ejemplo de MAT

0	0	0	0	0	0	0	0
0	0	<u>1</u>	0	0	0	0	0
0	0	0	0	0	<u>1</u>	0	0
1	1	1	1	1	1	1	0
1	1	1	1	1	<u>1</u>	1	0
1	1	<u>1</u>	<u>1</u>	1	1	1	0
1	1	1	1	1	1	0	0
(1,1)→1	1	1	1	1	1	0	0

x	y	r
3	3	2
3	7	0
4	3	2
6	4	1
6	6	0

Para una imagen $n \times n$, harán falta:

- $2\log_2 n$ bits para especificar las coordenadas de cada bloque centro.
- $\log_2 (n-1) - 1$ bits para especificar el radio.

así si hay b bloques en el MAT, el número total de bits necesarios para especificarlo es

$$b(2\log_2 n + \log_2 (n-1) - 1 + \log_2 (L-1)).$$

Y si $L=2$ (y n relativamente grande) se trata de, aproximadamente,

$$3b \log_2 n$$

Representación por bloques: **Árboles cuaternarios**

Asumimos por simplicidad que las imágenes S son binarias y tamaño $2^k \times 2^k$.

Método:

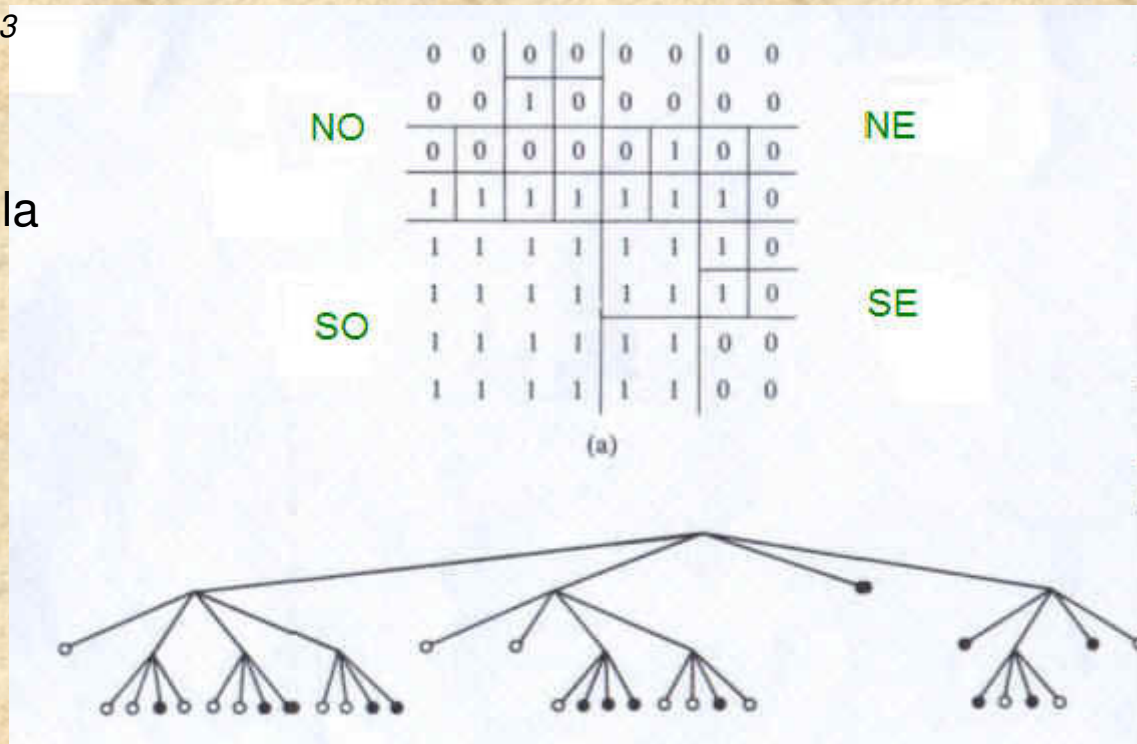
- El nodo raíz del árbol representa la imagen completa.
- Si la imagen tiene un sólo valor, marcamos el nodo raíz con ese valor y paramos. En otro caso, añadimos cuatro descendientes al nodo raíz, representando los cuatro cuadrantes de la imagen. Si un bloque tiene valor constante, su nodo es un nodo hoja; en otro caso, su nodo tiene cuatro descendientes correspondientes a los cuatro cuadrantes del bloque.
- El proceso se repite entonces para cada uno de esos nuevos nodos; así sucesivamente y como máximo k veces.
- Los nodos del nivel k , si hay alguno, son todos nodos hojas correspondientes a píxeles únicos. Los nodos de nivel h corresponden a bloques de $2^{k-h} \times 2^{k-h}$ píxeles.

Representación por bloques: Árboles cuaternarios

Ejemplo: imagen binaria $2^3 \times 2^3$

Árbol cuaternario de altura 3.

El orden de los hijos de cada fila es NO, NE, SO, SE.



- El espacio para almacenar el árbol es proporcional al número de nodos.
- No hay redundancia en cuanto a píxeles que aparezcan en dos nodos (como ocurría en MATs).
- Mal comportamiento respecto a traslaciones.

Representación mediante el borde de la imagen

Esta clase de representaciones hace uso del hecho de que cada uno de los conjuntos conexos de la imagen está determinado mediante la especificación de sus bordes.

Nos centraremos en imágenes **binarias**.

Dada una imagen binaria con la (p,q) -adyacencia (p -adyacencia para negros y q -adyacencia para blancos), el **borde** de la imagen (en negro) es el conjunto de píxeles negros con al menos 1 q -vecino blanco.

Representación mediante el borde de la imagen

Una **curva borde** de una imagen se determina especificando un punto de comienzo y una secuencia de movimientos alrededor del borde.

Introducimos dos formas de dar la secuencia de movimientos:

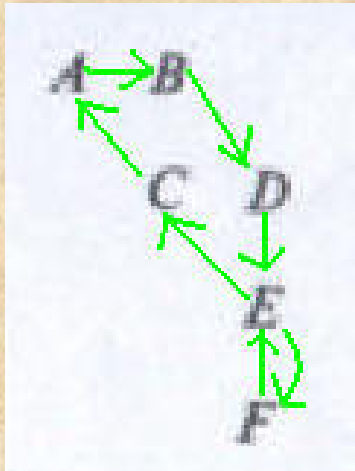
- Código de Cadenas.
- Código de Fisuras.

Representación mediante el borde de la imagen

Código de Cadenas.

Numeremos a los vecinos de **P** como sigue:

3	2	1
4	P	0
5	6	7



Por ejemplo, la secuencia de movimientos comenzando en A en esta figura es **0766233**.

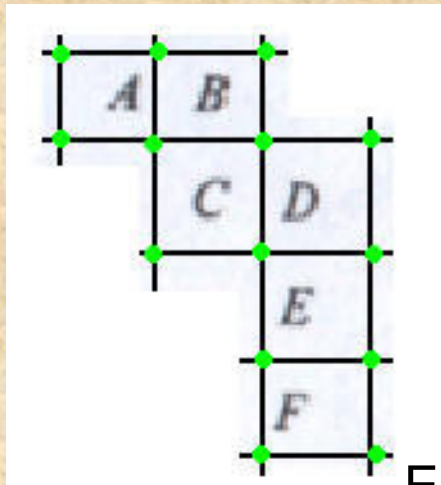
Este tipo de secuencias se denomina **código de cadenas**.

Un borde se define así dando las coordenadas de un punto de inicio junto con un código de cadenas representando una secuencia de movimientos

Representación mediante el borde de la imagen

Código de Fisuras.

Si seguimos las fisuras alrededor de un borde, entonces los movimientos son a izquierda, derecha, arriba o abajo; si denotamos la dirección 90° por i , estos movimientos pueden ser representados por una secuencia de números de dos bits (0,1,2,3).



1
2 0
3

Por ejemplo, la secuencia de movimientos comenzando en la esquina superior izquierda de A en esta figura es **00303332112121**.

Este tipo de secuencias se denomina **código de fisuras**.

Un borde se define así dando las coordenadas de una fisura de inicio junto con un código de fisuras representando una secuencia de movimientos

Representación mediante el borde

Codigo de Cadenas	Representado por números de 3 bits	Secuencias más cortas
Codigo de fisuras	Representado por números de 2 bits	Secuencias más largas

Código de cadenas diferencial:

Es una secuencia $(a_0, a_1, \dots, a_n)$ de los siguientes números: 0, ± 1 , ± 2 , ± 3 , 4 .
Estos **8** (*igual que el código de cadenas*) valores representan sucesivos cambios de dirección.

Por ejemplo, el 0 representa la no existencia de giro, ± 1 representa 45° de giro a la derecha o a la izquierda, ± 2 y ± 3 representan, de forma similar, giros de 90° y 135° , y 4 representa un giro de 180° . Hay valores muy comunes (ej. 0 y 1) y otros raros (ej. 4), es decir, no son equiprobables, por lo que puede usarse un código de longitud variable para eliminar redundancia.

Un borde es determinado mediante la especificación de las coordenadas del punto de inicio, la dirección inicial y el código de cadena diferencial.

El caso del código de fisura diferencial es análogo. Valores 0° y $\pm 90^\circ$.

Representación mediante el borde de la imagen

Para practicar:

Algoritmo de cálculo de código de fisuras

Obs.: El píxel que se encuentra más arriba-izquierda tiene de coordenadas (0,0)

Los movimientos vienen dados por la regla:

	3	
0		2
	1	

Algoritmo de búsqueda de bordes mediante código de fisuras.

Obs.: El píxel que se encuentra más arriba-izquierda tiene de coordenadas (1,1)

Los movimientos vienen dados por la regla:

	3	
2		0
	1	

Representaciones de los conjuntos derivados

Se entiende por conjuntos derivados de unas imágenes como aquellas que se obtienen a partir de unas dadas.

Ejemplo: La unión, intersección, complementario.

Objetivo: A partir de imágenes representadas del mismo modo (filas, bloques, bordes) obtener las representaciones de los conjuntos derivados.

Aquí, por simplicidad, sólo trataremos **imágenes binarias** S y T.

Nota: En el libro de Rosenfeld y Kak (Digital Picture Processing), hay una sección dedicada a algoritmos de conversión entre representaciones ([Capítulo 11](#)).

Representaciones de los conjuntos derivados

Operaciones lógicas en imágenes binarias:

- **Inverso de una imagen:** a cada píxel p aplicar la operación $\text{NOT}(p)$ donde $\text{NOT}(0)=1$ y $\text{NOT}(1)=0$.
- **Intersección de dos imágenes A y B:** $p(A) \text{ AND } p(B)$ donde el píxel de la intersección es negro si y sólo en ambas imágenes era negro.
- **Unión:** $p(A) \text{ OR } p(B)$ donde el píxel de la unión es negro si y sólo en alguna de las imágenes era negro.
- **Resta, B - A:** $\text{NOT}(A) \text{ AND } B$ donde un píxel es negro si era negro en B pero no lo era en A.
- **Unión menos la intersección (excluyente):** $A \text{ XOR } B$ donde un píxel es negro si lo era en A ó en B pero no en ambos.

Representaciones de los conjuntos derivados

Dada la representación de la longitud de la secuencia de S (y de T).

- La longitud de secuencia del **inverso de S** se obtiene simplemente invirtiendo la designación del primer valor de cada fila.
- Algoritmo para obtener L (lista de secuencia) la **intersección de S y T** a partir de sus respectivas listas L_1 y L_2 .

Si tanto L_1 como L_2 comienzan con secuencias de 1's, también lo hace L ; en otro caso, L comienza con 0.

- Ejemplo de matriz simple

```
101111111111100
000011101110000
000000111000000
000011101110000
001110000011100
```

- Ejemplo de correspondencia *runs*

```
1 : 1 1 11 2
0 : 4 3 1 3 4
0 : 6 3 6
0 : 4 3 1 3 4
0 : 2 3 5 3 2
```

Representaciones de los conjuntos derivados

- Algoritmo para obtener $L=c: c_1 c_2 c_3 \dots$, la lista de secuencia de la **intersección de S y T** a partir de sus respectivas listas:
 $S_1=a: a_1, a_2 a_3 \dots$ y $T_1=b: b_1 b_2 b_3 \dots$, siendo $a, b=0,1$
 1. Si tanto a como b *valen* 1, entonces c vale 1 y c_1 es el mínimo entre a_1 y b_1 .
 2. Si tanto a como b *valen* 0, entonces c vale 0 y c_1 es el máximo entre a_1 y b_1 .
 3. Si a vale 0 y b vale 1, entonces c vale 0 y c_1 es a_1 (análogamente, si a vale 1 y b vale 0).
- A continuación, rescribimos las nuevas secuencias S_1 y T_1 que surgen al eliminar los primero c_1 píxeles y volvemos a empezar para calcular c_2 .

- Ejemplo de matriz simple

```
101111111111100
000011101110000
000000111000000
000011101110000
001110000011100
```

- Ejemplo de correspondencia *runs*

```
1 : 1 1 11 2
0 : 4 3 1 3 4
0 : 6 3 6
0 : 4 3 1 3 4
0 : 2 3 5 3 2
```

Representaciones de los conjuntos derivados

Ejercicios:

- a) Describir un algoritmo para obtener la unión de dos imágenes usando la representación MAT.

0	0	0	0	0	0	0	0
0	0	<u>1</u>	0	0	0	0	0
0	0	0	0	0	<u>1</u>	0	0
1	1	1	1	1	1	1	0
1	1	1	1	1	<u>1</u>	1	0
1	1	<u>1</u>	<u>1</u>	1	1	1	0
1	1	1	1	1	1	0	0
(1,1)→1	1	1	1	1	1	0	0

x	y	r
3	3	2
3	7	0
4	3	2
6	4	1
6	6	0

- b) Idem para describir el árbol cuaternario de la inversa de una imagen.
- c) Idem para describir los códigos de fisura de la inversa de una imagen

Codificación en planos de bits de una imagen en escala de grises

Un método sencillo para la descomposición de una imagen en escala de grises de L bits en un conjunto de imágenes binarias consiste en separar los L dígitos de cada píxel y formar L imágenes de 1 bit cada píxel (planos de bits).

Es decir, si el nivel de gris asociado a un píxel es

$$a_{L-1} 2^{L-1} + a_{L-2} 2^{L-2} + \cdots + a_1 2^1 + a_0 2^0$$

se toma cada uno de los coeficientes a_k para formar el k-ésimo plano de bits, $k=0, \dots, L-1$.

Después, cada una de las imágenes binarias puede ser representada en alguna de las formas descritas anteriormente.

Inconveniente: pequeñas variaciones en los niveles de gris tienen un impacto significativo en la complejidad de los planos de bits.

Por ejemplo, si en la imagen inicial hay un píxel de intensidad 127 (0111111) junto a otro de 128 (1000000), en todos los mapas de bits correspondientes habrá un salto de 0 a 1 (ó de 1 a 0).

Codificación en planos de bits de una imagen en escala de grises

Código de Gray

Un método de descomposición alternativo (que reduce el efecto de pequeñas variaciones de niveles de gris) consiste en representar primero la imagen mediante un *código de Gray* de L bits.

- Si los dígitos en binario del número son

$$a_{L-1}a_{L-2}\dots a_1a_0$$

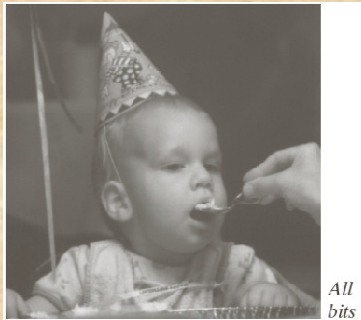
- El mismo número en código de Gray es

$$g_{L-1}g_{L-2}\dots g_1g_0$$

donde $g_i = a_i + a_{i+1} \bmod 2$ si $i < L-1$ y $g_{L-1} = a_{L-1}$

Número	Código binario	Código Gray
0	0000	0000
1	0001	0001
2	0010	0011
3	0011	0010
4	0100	0110
5	0101	0111
6	0110	0101
7	0111	0100
8	1000	1100

Codificación en planos de bits de una imagen en escala de grises



Los cuatro planos de bits más significativos de esta imagen, correspondientes a los coeficientes a_7 , a_6 , a_5 , a_4 . La segunda fila de imágenes corresponde a los planos de bits usando el código de Gray.

