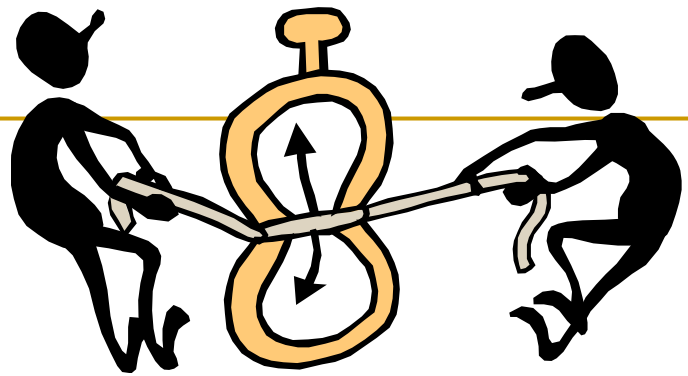
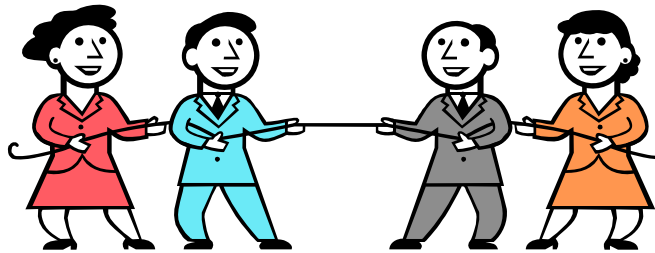
Algo sobre Compresión de Datos

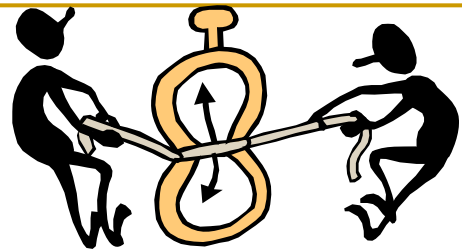


Un conflicto histórico



- Algoritmos tradicionalmente diseñados para ahorrar tiempo…..y espacio
- Posibles conflictos:
  - Estructuras que son más eficientes en términos de tiempo que requieren más espacio
  - Estructuras que son eficientes en términos de espacio pero que requieren más tiempo de procesamiento
- A veces no hay conflicto:
  - Guardar la información en forma tal que ocupe el menor espacio posible para que su transmisión lleve el menor tiempo posible
- Aunque cierto conflicto subsiste:
  - Si la información está muy bien compactada, puede requerir tiempo de "descompactación"
- Argumento: "No hace falta comprimir porque ahora los dispositivos de almacenamiento son muy baratos"
- Contraargumentos:
  - "Al manejar grandes volúmenes de información, los ahorros son más importantes"
  - "Time is money"

Compresión de archivos



- En general, las técnicas de compresión de archivos aprovechan la redundancia contenida en los archivos

- Varios métodos:
  - Codificación por longitud de series
  - Códigos de Huffman
  - Familia de métodos ZL (Ziv-Lempel)

Codificación por longitud de series

- Empecemos por un chiste :

Un Sr. llega a la casa y encuentra una nota de X:

“Pasó un Sr., dijo que era urgentísimo que lo llame sin falta al “77774444442222222999999”

Intrigado, mira la nota al derecho y al revés sin entender a qué número tiene que llamar. Definitivamente ese no es un número de teléfono. Finalmente, llama a X y se lo dice

- ¿Cómo que no? Yo escribí lo que me dictaron:

“cuatro siete seis cuatro ocho dos cinco nueve”

- La codificación por longitud de series es así….pero al revés!
- Originalmente, era un chiste de gallegos. X podría ser la esposa, el esposo, un gallego, el portero, la nueva secretaria, pero en todos los casos me acusarían de falta de correctitud política, así que lo dejamos como X.

Fundamentos

- La redundancia más simple que se puede encontrar son las largas series de caracteres repetidos
- Ej:
  - AAAABBBAABBBBBCCCCCCCCCDABCBAAABBBBCCCD
- Reemplazamos cada repetición por un sólo ejemplar del caracter repetido acompañado por la cantidad de repeticiones:
  - 4A3B2A5B8C1D1A1B1C1B3A4B3C1D
- O mejor todavía
  - 4A3BAA5B8CDABCB3A4B3CD

Variante para archivos binarios

- Almacenar simplemente la longitud de cada secuencia, sabiendo que los caracteres serán alternantes
- Es útil si hay pocas series cortas.

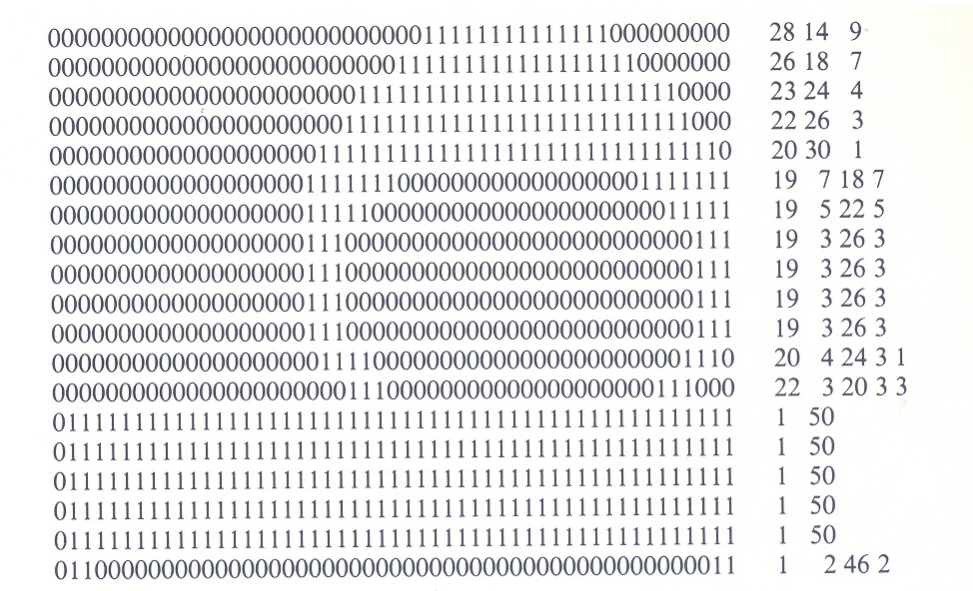


Figura 22.1 Una matriz de puntos típica, con información de la codificación por longitud de series.

- Notar que, sabiendo que la longitud de cada línea es de 51 pixels, no hace falta almacenar “fines de línea”
- Si se utilizan 6 bits para cada longitud, se necesitan 384 bits contra los 975 de la representación explícita.

Volvamos a archivos genéricos

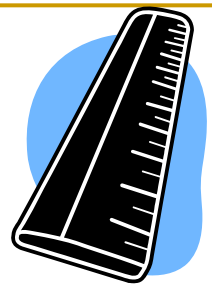
- Necesitamos representaciones separadas para los elementos del archivo y los de su versión codificada (en el chiste, si no tuvieramos los espacios, como distinguiríamos “47 64” de “4 764” o de “476 4”?
- Supongamos que tenemos sólo las 26 letras y el espacio en blanco, ¿Cómo hacemos?
- 1ra posiblidad: usar un caracter que no aparece como *caracter de escape*. Cada aparición indica que las siguientes son un par (longitud,carácter), y representamos una longitud i con la i-ésima letra
- Ej. (usando la “Q” como escape”)
  - QDABBBAAQEBQHCDABCBAAAQDBCCCD
- Notar que no codificamos secuencias de menos de 4 caracteres (ya que necesitamos al menos 3 para codificar! Esos 3 se llaman “secuencia de escape”)

Seguimos con archivos genéricos

- Problema: ¿qué hacemos si el caracter de escape sí aparece?
- Posible solución: representarlo con una secuencia de escape “imposible”: Q<espacio> (sería imposible si el <espacio> corresponde al número 0)
- ¿Qué pasa si tratamos de comprimir un archivo ya comprimido?
- ¿cómo hacemos para series más largas que 26 elementos?
  - podemos usar varias secuencias de escape: QZAQYA significa: “51 Aes” (26+25)
  - Otra posibilidad: si se esperan secuencias largas, utilizar más de un caracter para codificar longitudes.
- Cierre: este método no es muy útil para archivos de texto, en los que los únicos caracteres que aparecen en largas secuencias son los <espacio>.

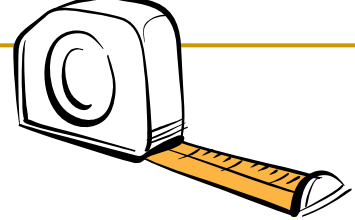
Códigos de Huffman

Códigos de longitud fija



- Ejemplo:
  - ABRACADABRA
- Representando cada letra con 5 bits, arrancando con el 00001 para la “A” etc.:
  - 00001000101001000001000110000100100000010001010 01000001
- Decodificación fácil: leer de a 5 bits y reemplazarlo por la letra correspondiente

Códigos de longitud variable



- Idea: ¿por qué usar la misma cantidad de bits para representar caracteres muy frecuentes que para caracteres muy raros?
- Por ejemplo, qué pasaría si usaramos la siguiente tabla:

A	0
B	1
R	01
C	10
D	11

- ABRACADABRA = 0 1 01 0 10 0 11 0 1 01 0
- (15 bits en lugar de 55)
- Mentira: ¿qué pasa con los <espacio>?
- ¡Igual es más corto!
- En realidad…..todavía es peor: hay que incluir la tabla en el texto comprimido (pero para textos grandes, eso no molesta mucho…)

Códigos Prefijos

- ¿Podríamos evitar la necesidad de enviar separadores?
- ¡Sí!: Códigos Prefijos – el código de ningún caracter es prefijo del código de otro.
- Ej.:

A	11
B	00
R	011
C	010
D	10

- Hay una única forma de decodificar el mensaje
  - 11000111101011101100011111
- ¡Hay una única forma de decodificar cualquier mensaje!

Representación de códigos prefijos

- Podemos usar un TRIE para representar el código.
- De hecho, cualquier trie con M nodos externos puede ser utilizado para codificar cualquier mensaje con M caracteres diferentes: el código de cada caracter se determina por el camino de la raíz al caracter.

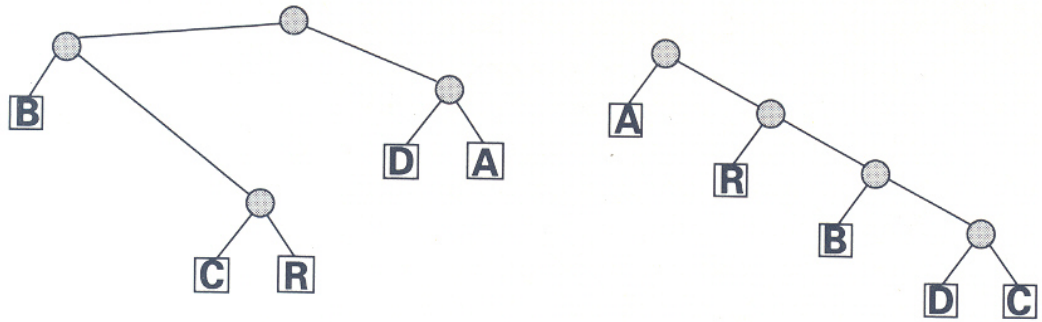


Figura 22.2 Dos tries de codificación para A, B, C, D y R.

- Con el código de la derecha, la cadena sería 01101001111011100110100, que es más corta
- ¿por qué?

Códigos prefijos óptimos

- Código de Huffman (1952)
- Dado el mensaje, calcular la frecuencia de aparición de cada caracter.
- Ej.
  - UNA CADENA SENCILLA A CODIFICAR CON UN NUMERO MINIMO DE BITS

CAR	b	A	B	C	D	E	F	I	L	M	N	O	R	S	T	U
FREC	11	6	1	5	3	4	1	6	2	3	7	4	2	2	1	3

Construcción del TRIE

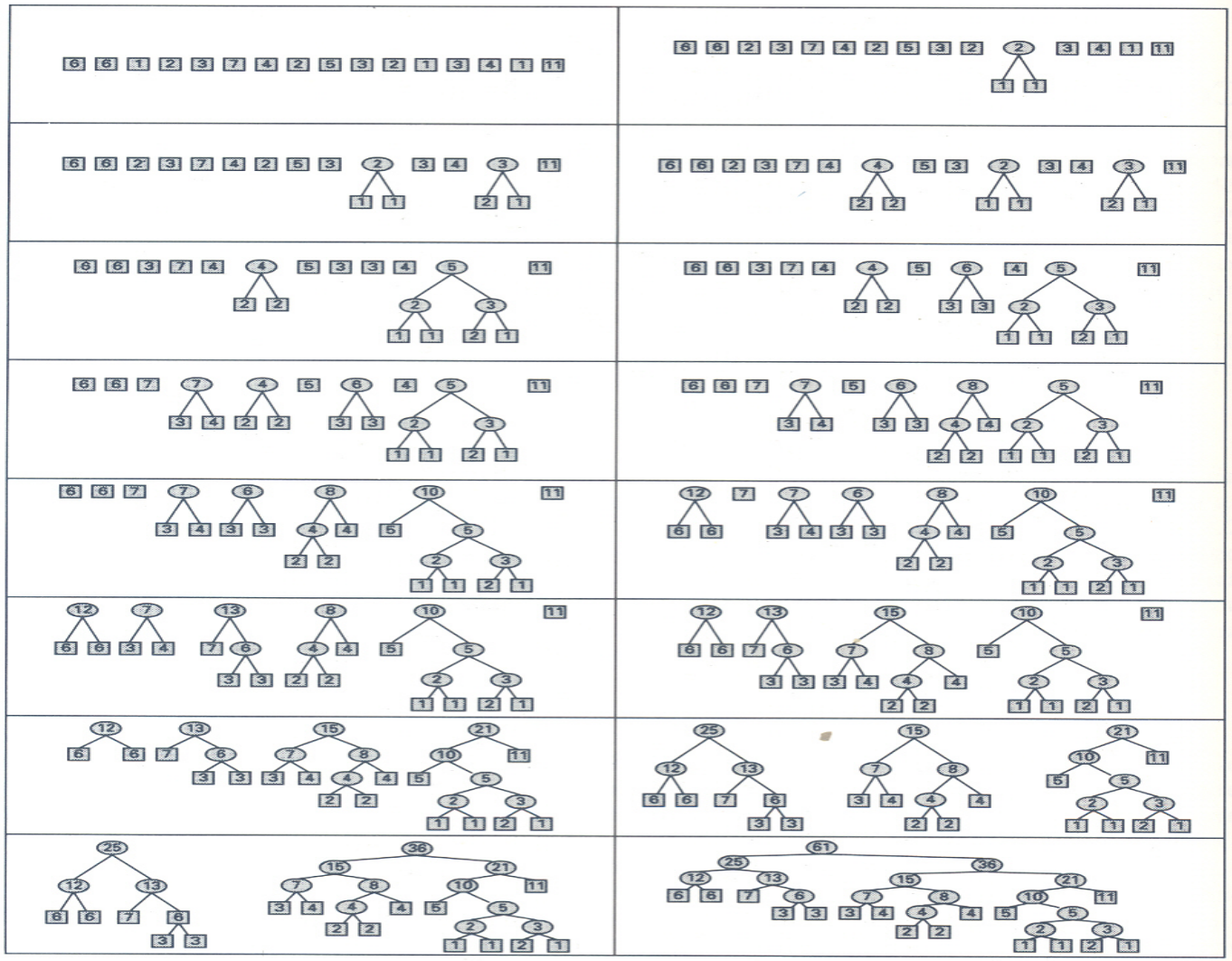


Figura 22.4 Construcción de un árbol de Huffman.

Construcción del TRIE (cont.)

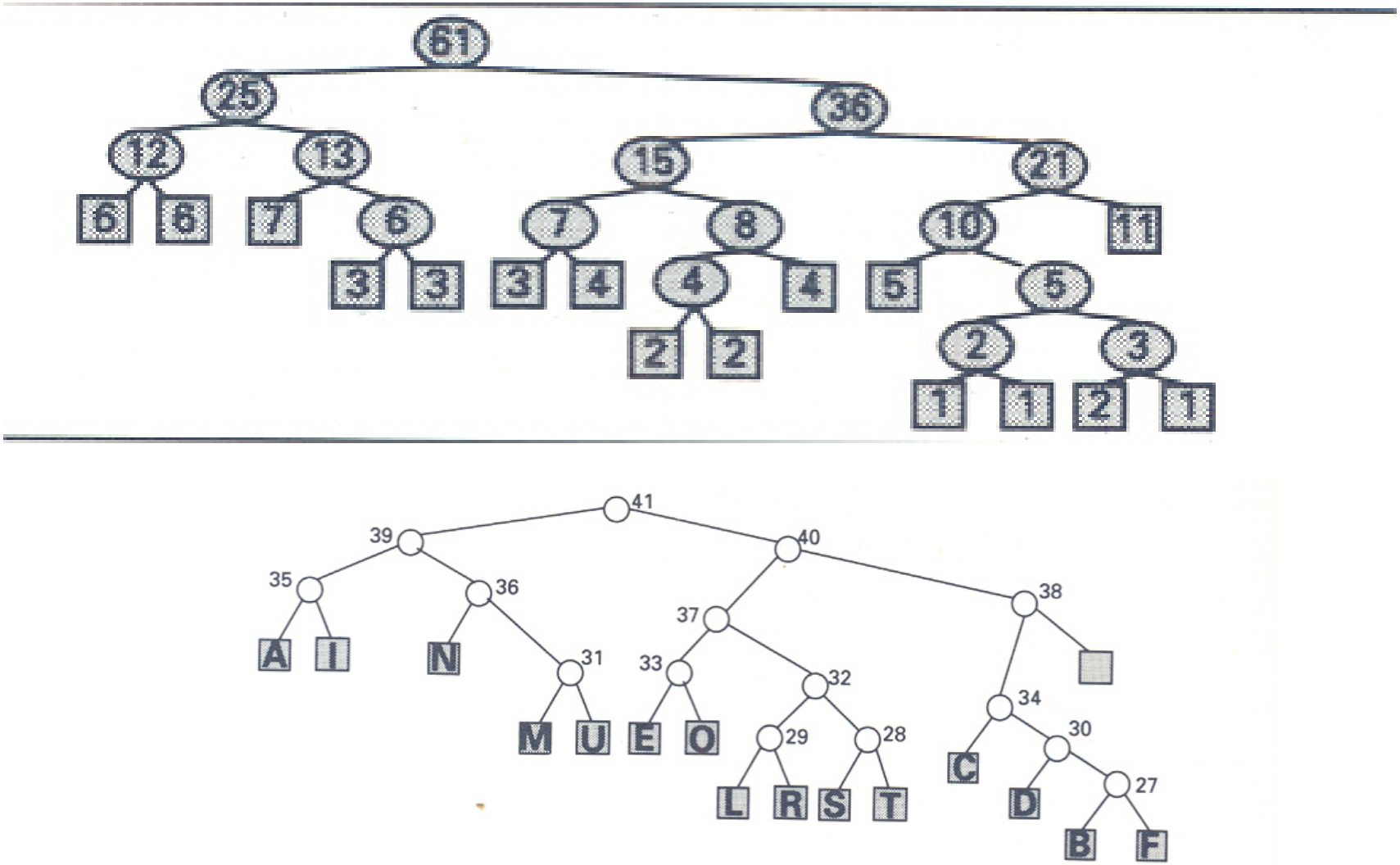


Figura 22.5 Un trie del código de Huffman para UNA CADENA SENCILLA...

Atención: ambas figuras no representan el mismo código

Volviendo a nuestro ejemplo

- Se codifica con la siguiente secuencia
  - 011101000011111000001101010000100001111011010000101100001101001010000011100011111001001111010001110111001110000010101111110010010101110111010111010011101101000101011001111011000101000101101001111110101001111101100011

- Esta secuencia tiene 228 bits en lugar de los 300 que se utilizarían en la codificación directa.

Optimalidad del Código H

- ¿Por qué este código es el mejor?
- Definición: La longitud ponderada del camino externo de un árbol es la suma, para todos los nodos externos, del producto de su “peso” por la distancia a la raíz.
- Propiedad: La longitud del mensaje codificado es igual a la longitud ponderada del camino externo del árbol de frecuencias de Huffman

Optimalidad del Código H

- Teorema: Ningún árbol con las mismas frecuencias en los nodos externos puede tener una longitud ponderada del camino externo menor a la del árbol de Huffman.
- Lema: Dado un código, podemos reacomodar las hojas del árbol para que los dos caracteres menos frecuentes sean “hermanos” y obtener un código mejor o igual.
- Dem:
  - El menos frecuente (x) es la hoja más lejana (si no lo es, hago un intercambio y obtengo un código mejor o igual).
  - El padre de x tiene otra hoja hija h (si no la tiene, por qué x no está más arriba?)
  - h tambiénes una hoja más lejana. Si está ocupada por un nodo que no es el segundo menos frecuente, sino por uno más frecuente, podemos hacer un intercambio que mejorará el código.

Optimalidad del Código H

- Dem del teorema – Por inducción en la cantidad de caracteres del alfabeto.
  - Base: un caracter, trivial
  - Paso inductivo:
  - Sea T el árbol construido por el algoritmo de Huffman sobre todo el alfabeto.
  - Sean x,y los caracteres menos frecuentes, f(x), f(y) sus frecuencias
  - Sea el árbol T´construido por el algoritmo de Huffman sobre el alfabeto en el que se eliminan x e y reemplazándolos por un nuevo caracter z, con frecuencia f(z)=f(x)+f(y).
  - Por hipótesis inductiva, T´ es óptimo.
  - LPCE(T)=LPCE(T´)+f(x)+f(y).
  - Supongamos existe T´´ con LPCE(T´´)<LPCE(T). En T´´ x e y son hermanos.
  - Pero entonces sea T´´´=T´´ reemplazando al padre de x e y por z con f(z)=f(x)+f(y)
  - Tenemos que

$$LPCE(T''') = LPCE(T'') - f(x) - f(y) < LPCE(T) - f(x) - f(y) = LPCE(T')$$

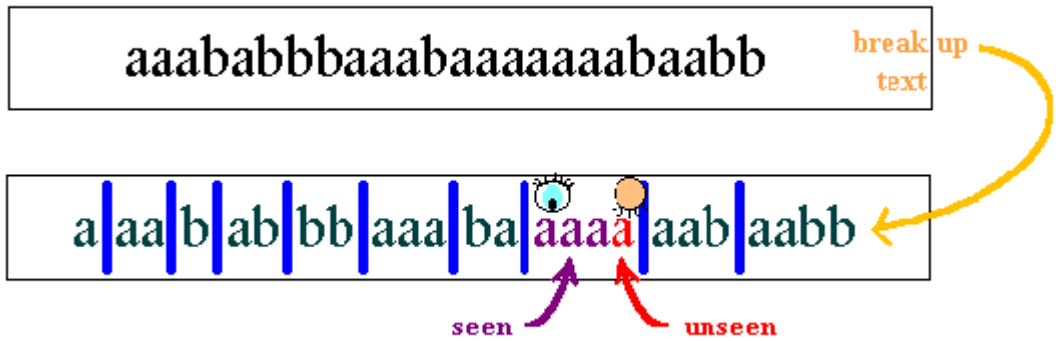
  - Absurdo, pues supusimos T´ óptimo.

Familia de métodos ZL (¡Sólo para tener idea!)

- Propuestos por Abraham Lempel y Jacob Ziv, en 1977
- También basados en la repetición
- Pero de patrones más complejos
- No necesita ni presuponer frecuencias ni hacer dos pasadas
- También utiliza tries
- Reemplaza secuencias completas por un puntero a una posición del TRIE
- Utilizada en los conocidos métodos GZip, etc.

Mini-ejemplito ZL

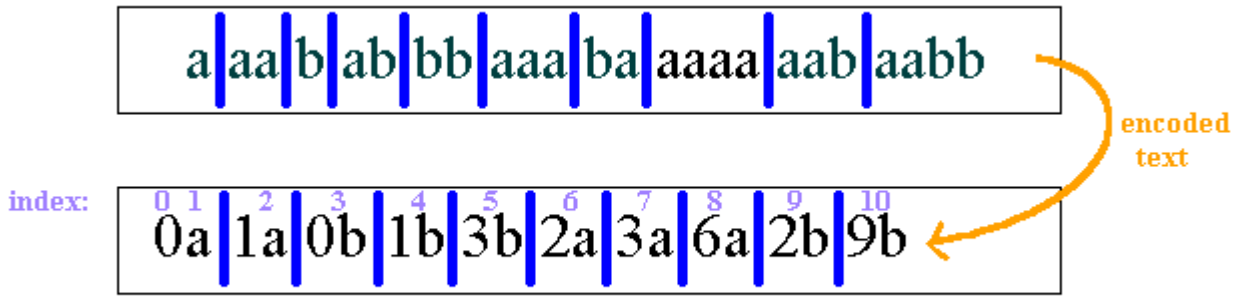
- Ej.



- Partimos el texto en bloques "ya visto + nuevo carácter"
- Luego, numeramos los bloques

Mini-ejemplito ZL (sigue)

- Reemplazamos cada bloque por un puntero al bloque ya visto y el nuevo carácter



- Podemos pensar todo como un TRIE
- ¿Cómo sigue este?