TEORÍA DE LA INFORMACIÓN

Tratamiento probabilístico de la Información

Compresión de datos

continuación

INGENIERÍA DE SISTEMAS

Cursada 2016

MÉTODOS DE COMPRESIÓN

SIN PÉRDIDA	CON PÉRDIDA
• Métodos estadísticos (procesan un símbolo por vez teniendo en cuenta sus probabilidades): ○ Shannon, Fano ○ Huffman estático y semiestático ○ Huffman dinámico (FGK) ○ Codificación aritmética (genera un número para todo el mensaje) • Basados en diccionario o sustitucionales: ○ Algoritmos LZ (codifican referencias a cadenas de símbolos previos): LZ77, LZ78 y sus variantes • Run-Length (codifican secuencias de símbolos coincidentes)	• JPEG • Run-Lenght c/pérdida • Compresión Fractal • VQ (Cuantificación vectorial) • MPEG • …

CODIFICACIÓN ARITMÉTICA

Elías-Abramson (1963), Witten y otros (1987)

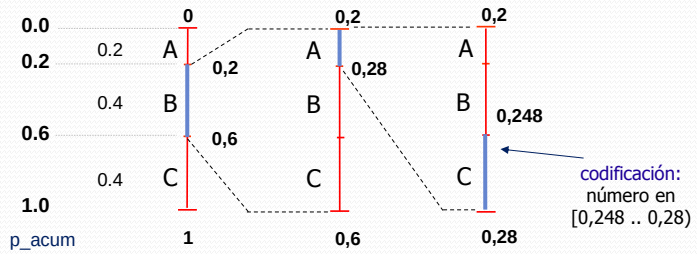
- Procesa un símbolo por vez pero…
- el mensaje completo se codifica mediante un número de alta precisión

```
lim_inf= 0.0,  lim_sup= 1.0
while ( (s= leer(simbolo) ) != EOF )
{   rango = lim_sup - lim_inf
    lim_inf = lim_inf + rango* p_acum_inf( s )
    lim_sup = lim_inf + rango* p_acum_sup( s )
}
Cod → cualquier número en [lim_inf, lim_sup)
```

Ejemplo: codificar el mensaje 'B A C' con p(A)=0.2, p(B)= 0.4, p(B)= 0.4

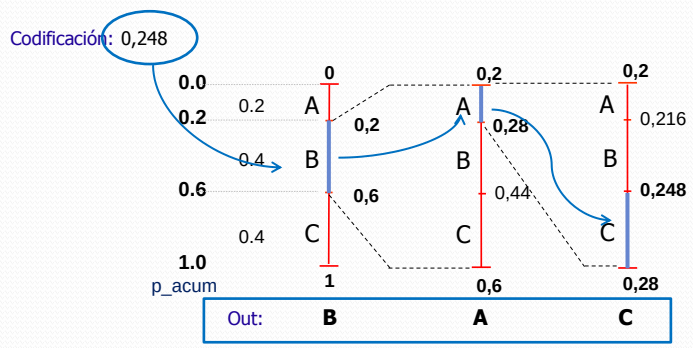
NOTA:

Es irrelevante el orden en que se consideren los símbolos en el rango, mientras compresor y descompresor lo hagan igual



CODIFICACIÓN ARITMÉTICA

El decodificador funciona de manera similar (alg. simétrico) → determinar en qué rango se encuentra el número recibido en cada paso de contracción del intervalo



Cuándo termina el proceso de descompresión?

→ incluir un símbolo de fin de mensaje (de baja probabilidad)

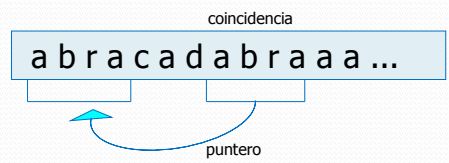
→ o anteponer la longitud del mensaje (en esquemas semi-estáticos o es conocido)

CODIFICACIÓN ARITMÉTICA

- La implementación requiere ajustes sobre la formulación original:
  - Requeriría aritmética de precisión arbitraria (debido a las sucesivas operaciones de contracción del intervalo) → usar aritmética entera y controlar posibles situaciones de *overflow* y *underflow*
  - No se podría enviar el mensaje hasta completar la codificación → transmisión incremental (si los límites del rango coinciden se van transmitiendo decimales)
- Generalmente logra mejor resultado que el método de Huffman si las probabilidades son desbalanceadas y si el mensaje es pequeño, pero con mayor costo computacional (no logra superarlo para fuentes con prob. ~1/2ª, sin importar la longitud)
- La versión adaptativa es mucho más simple y menos costosa que el algoritmo FGK (Huffman dinámico)
- También forma parte de la especificación de estándares de compresión

CODIFICACIÓN DE DICCIONARIO

- La codificación consiste en el reemplazo de cada "frase" en el mensaje (grupo de símbolos de longitud variable) por un índice o puntero a un diccionario de frases
- El diccionario se conforma por las frases previamente encontradas
- Los esquemas utilizados consideran diccionarios dinámicos o adaptativos (se incorporan nuevas frases al diccionario a medida que la compresión procede)

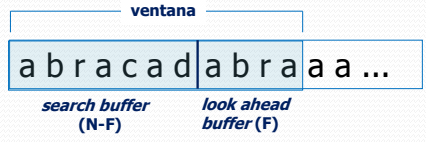


- Compresión → la representación de los índices ocupa menos espacio que las frases (cuanto mayor la coincidencia de símbolos en la frase → mayor compresión)
- El compresor es más costoso por la búsqueda de coincidencias, el descompresor sólo debe recuperar las cadenas indicadas por los punteros → alg. asimétrico

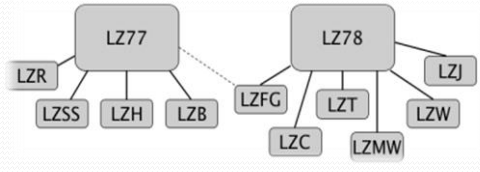
ALGORITMOS LZ

Lempel, Ziv (1977 y 1978)

- Plantean una "familia de algoritmos" de diccionario adaptativo → el diccionario se construye mientras se codifica, a partir del mensaje ya procesado
- Se conforma una **ventana** de búsqueda que contiene la secuencia de símbolos a codificar (*lookahead buffer)* y la ya procesada (*search buffer)*

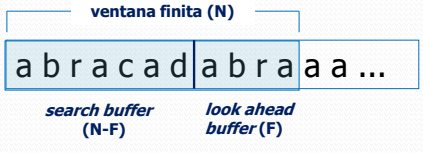


- Opciones:
  - LZ77 → Ventana finita (deslizante)
  - LZ78 → Ventana infinita
- Distintas variantes (implementación)



ALGORITMO LZ77

Algoritmo de compresión LZ77:



Puntero: (I , L , D)

- I → posición de Inicio
- L → longitud
- D → próximo símbolo (no coincidente)

```
while (look ahead buffer no vacío) {
  generar puntero (posicion, long) a la mayor coincidencia
  if (long >0) {
    generar_salida(posicion, long, proximo_simbolo)
    adelantar la ventana long+1 posiciones
   }else {
    generar_salida(0, 0, proximo_simbolo)
    adelantar la ventana 1 posicion
  }
}
```

ALGORITMO LZ77

EJEMPLO

Codificación:

(N=12 F=6)

triplas (inicio, long, próximo)

6 5 4 3 2 1 0			
	A B A C	In: **A**	Out: (0,0,A)
A	B A C	In: **B**	Out: (0,0,B)
A B	A C	In: **AC**	Out: (2,1,C)
A B A C			

Decodificación:

	6 5 4 3 2 1 0		
In: (0,0,A)			Out: **A**
In: (0,0,B)	A		Out: **B**
In: (2,1,C)	A B		Out: **AC**
	A B A C		

ALGORITMO LZ78

Algoritmo de compresión LZ78 (propuesta original):

- Usa ventana "infinita" (todo el mensaje procesado previamente)
- Mantiene un diccionario explícito
- Genera un par (N , D) y agrega N D como nueva entrada al diccionario

N: puntero a máxima coincidencia

D: próximo símbolo (no coincidente)

Una mejora: **LZW** (versión más utilizada):

- Evita codificar el "próximo símbolo no coincidente" (inicia la sig. frase)
- Precarga el diccionario con los símbolos individuales
- Genera un índice al diccionario y agrega la frase codificada + 1º símbolo no coindente

ALGORITMO LZW

Algoritmo de compresión
(publicado por Welch, 1984)

```
w= ' '
while (entrada no vacía){
  k= siguiente_simbolo
  if (wk está en el diccionario){
    w= wk
   }else {
    generar_indice(w)
    agregar wk al diccionario
    w= k
   }
  }
```

Propuesta original: usa puntero de 12 bits → hasta 4096 entradas al diccionario

Cuando se completa → seguir con diccionario estático / eliminar frases menos frecuentes / vaciar y reiniciar

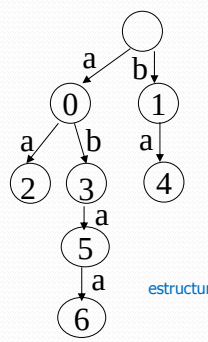
ALGORITMO LZW

EJEMPLO

Codificación:

a a b a b a b a a a

Out: 0 0 1 3 5 2



estructura de árbol

Descompresión: leer número de frase y:

1. si está en el diccionario → decodificar y agregar *salida anterior + 1º símbolo de la salida actual*
2. si no está → decodificar y agregar *salida anterior + 1º símbolo de la salida anterior*

Se debe a que el codificador conoce el siguiente símbolo (el decodificador aún no!)

0 0 1 3 5 2

entrada	salida	agrega	
		0 a	*pre-cargadas*
		1 b	
0	a		
0	a	2 a a	
1	b	3 a b	
3	a b	4 b a	
5 (caso 2)	a b a	5 a b a	
2	a a	6 a b a a	

Lista de entradas (diccionario explícito)

BIBLIOGRAFÍA

Cover T., Thomas J., ***Elements of Information Theory,*** 2nd ed., John Wiley & Sons, 2006

Nelson M., Gailly J., ***The Data Compression Book,*** 2nd ed., M&T Books,1996

Bell T., Cleary J., Witten I., ***Text Compression*****,** Prentice Hall, 1990

Gonzalez R., Woods R., ***Digital Image Processing,*** 2nd ed., Prentice Hall, 2002