Facultad de Ciencias Exactas
UNIVERSIDAD NACIONAL DEL CENTRO DE LA PROVINCIA DE BUENOS AIRES

TEORÍA DE LA INFORMACIÓN

Tratamiento probabilístico de la Información

Compresión de datos

INGENIERÍA DE SISTEMAS

Cursada 2017

RUN LENGHT CODING (RLC)

- Se codifican secuencias de símbolos consecutivos iguales con un par *Símbolo, Repeticiones (longitud de la secuencia)* → utiliza información de contexto
- Se debe fijar el tamaño de la representación de ambos valores

Ejemplo 1:

In: A A A A A A A A B B B C C C C C (ORIGINAL → 16 bytes)

Out: A 8 B 3 C 5 (COMPRIMIDO → 6 bytes ☺) tasa 2,66:1 Suponiendo 1 byte para símbolo y 1 byte para longitud

Ejemplo 2:

In: A B C A B C B A B A C A A A A A (ORIGINAL → 16 bytes)

Out: A 1 B 1 C 1 A 1 B 1 C 1 B 1 A 1 B 1 A 1 C 1 A 5 (COMPRIMIDO → 24 bytes ☹)

- Mejora: fijar un número mínimo de repeticiones para representar una secuencia como un par *S R* y anteponer un bit *flag* para indicar si sigue un par codificado o un símbolo sin codificar (según sea conveniente).
- Aplicable en datos que contienen largas repeticiones de símbolos (ejemplo: gráficos con fondo y objetos de color plano)
- Forma parte de estándares de compresión (ej: JPEG, MPEG, codif. Fax ITU-T T4 Group 3)

RUN LENGHT CODING (caso BINARIO)

- En imágenes Blanco/Negro no se codifica el tono (sería redundante, ya que es una secuencia de '0' y '1' alternados)
- Por convención la longitud inicial siempre es para el símbolo '0' (negro)

Ejemplo 1:

In: 0 0 0 0 0 0 0 0 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 .....

Out: 8 5 9 7 .....

Ejemplo 2:

In: 1 1 1 0 0 0 0 0 .....

Out: 0 3 5 .....

- En este caso tener en cuenta que las imágenes B/N asignan un bit por tono

RLC CON PÉRDIDA

Adaptación de *Run Lenght Coding* :

- Se considera una cierta diferencia (tolerancia T) en la intensidad de los píxeles de cada secuencia codificada
- La codificación es una secuencia de pares (v, r) donde:

  v= valor del primer símbolo de la secuencia
  r = cantidad de símbolos consecutivos con valor entre v–T y v+T

Ejemplo (con T= 5):

Secuencia original: 10 10 8 10 12 8 8 9 54 54 55 56 50 50 120 123 121 119 120

Codificación: 10 8 54 6 120 5

Decodificación: 10 10 10 10 10 10 10 10 54 54 54 54 54 54 120 120 120 120 120

Error: $e_{abs} \approx 1{,}31$

CODIFICACIÓN ARITMÉTICA

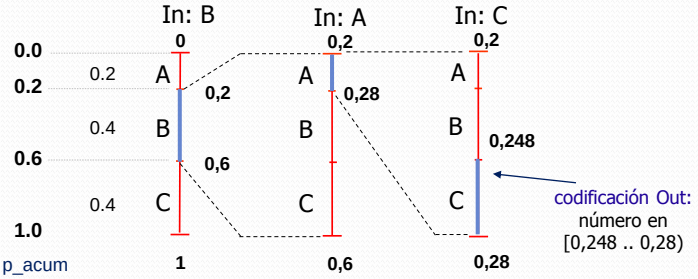
Elías-Abramson (1963), Witten y otros (1987)

- 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)
```

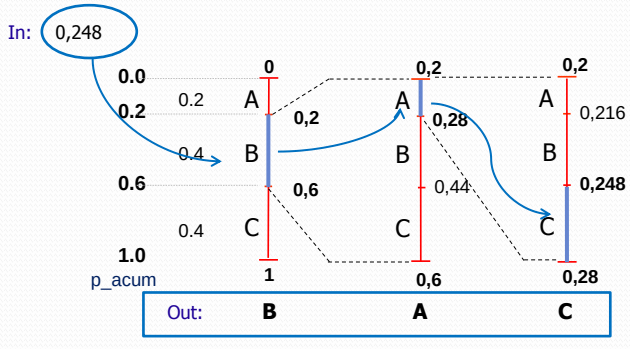
Ejemplo: modelo estático con p(A)=0.2, p(B)= 0.4, p(B)= 0.4
codificar el mensaje `B A C'

Nota:
Es irrelevante el orden de los símbolos en el rango, mientras el compresor y el descompresor lo hagan igual



DECODIFICACIÓN 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)

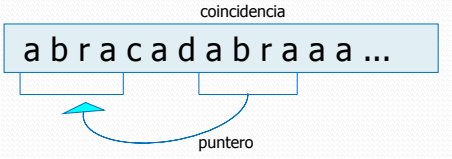
→ o anteponer la longitud del mensaje (en esquemas semi-estáticos o es conocido)
(Ej. anterior codificaría **3 0.248**)

CODIFICACIÓN ARITMÉTICA

- La implementación requiere ajustes sobre la formulación original:
  - Requiere aritmética de precisión → se usa aritmética entera con control de posibles situaciones de *overflow* y *underflow*
  - No se puede enviar la codificación hasta completar el mensaje → se usa 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

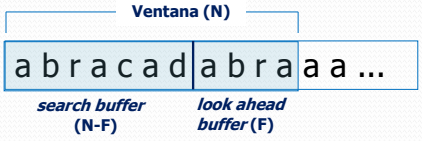


- Compresión → la representación de los índices ocupa menos espacio que las frases
- El compresor es más costoso por la búsqueda de coincidencias, el descompresor sólo debe recuperar las cadenas indicadas por los punteros → alg. asimétrico

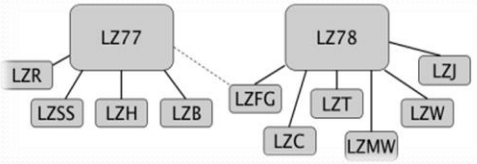
ALGORITMOS LZ

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)*



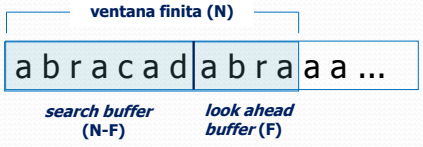
- Opciones:
  - LZ77 → Ventana finita (deslizante)
  - LZ78 → Ventana infinita
- Distintas variantes (implementación)

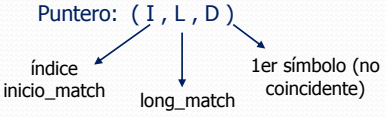


- Utilizan variantes de LZ: compress (Unix), pkzip, arj, gif, tiff, Acrobat Reader.

ALGORITMO LZ77

Algoritmo de compresión LZ77:





1. Se busca el *matching* de símbolos en el buffer *look ahead* en la ventana *search buffer* (símbolos ya procesados) y se codifica la 3-upla:

   (I, L, D)

2. Luego de la emisión de la 3-upla se desplaza la ventana L+1 símbolos.
3. Volver al paso 1, sino termino de leer toda la entrada

ALGORITMO LZ77

EJEMPLO

Codificación:

(N=12 F=6)

3-upla (inicio_mach, long_mach, 1er símbolo)

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 , D): N = puntero a máxima coincidencia; D = próximo símbolo (no coincidente)

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

- LZW: Es una variante de LZ78 publicada por Welch 1984
- Evita codificar el "próximo símbolo no coincidente" (inicia la sig. frase)

ALGORITMO LZW

Algoritmo de compresión LZW
(publicado por Welch, 1984)

1) Precarga el diccionario con los símbolos individuales
2) Lee la ***frase*** (con mayor coincidencia en diccionario) y la codifica con su índice
3) Agrega al diccionario una nueva entrada
   frase + 1º símbolo no coincidente
4) Si no termina el mensaje, vuelve al 2)

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

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

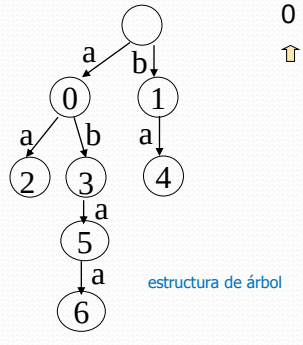
ALGORITMO LZW

EJEMPLO

Codificación:

In: a a b a b a b a a a

Out: 0 0 1 3 5 2



estructura de árbol

Decodificación: leer número de frase y:
1. si está en el diccionario → decodificar su frase (*salida actual*), 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*

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