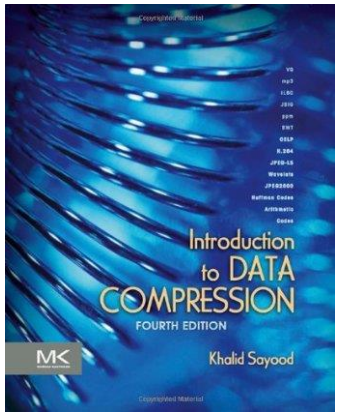
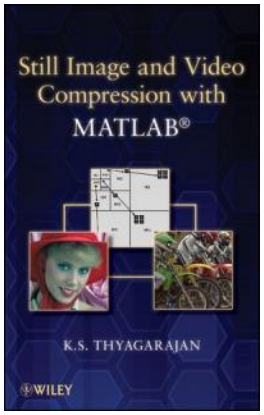
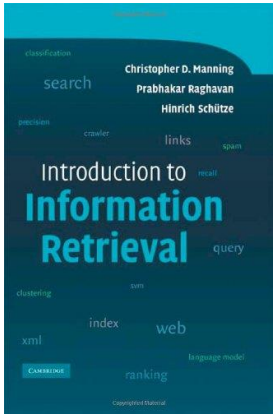
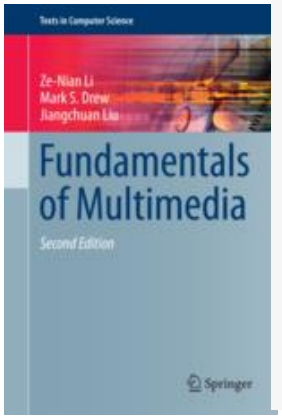








Codificación Basada en Diccionarios

Rafael Molina

Depto Ciencias de la Computación

e Inteligencia Artificial

Universidad de Granada

Contenidos

I. Objetivos del tema

1. En los temas anteriores examinamos métodos de codificación que suponían que la fuente generaba secuencias de símbolos independientes que habían, en algunos casos, sido decorreladas (calculando por ejemplo diferencias, recuerda nuestro bloque hacer algo).
2. Para aumentar la compresión intentaremos incorporar ahora más estructura de los datos en el proceso de compresión. Las técnicas que veremos, estáticas y dinámicas, construyen una lista de patrones frecuentes y transmiten el índice de la lista.
3. Estos métodos son especialmente útiles cuando el número de patrones es reducido (por ejemplo, en texto o en órdenes del computador) y aparecen frecuentemente.
4. Veremos algunos compresores y formatos de imágenes que utilizan LZW.

II. Introducción.

En muchos problemas de compresión la salida de la fuente es un patrón recurrente.

Una aproximación razonable a la codificación de este tipo de fuentes es la existencia de una lista o *diccionario* de los patrones más frecuentes.

Consideremos el alfabeto inglés formado por letras en minúscula (26) y signos de puntuación (6): , . ! ? : ; . En total 32 caracteres.

Si todos son igualmente probables necesitamos 5 bits para cada uno. Si formamos **palabras de cuatro caracteres**, tenemos $32^4=2^{20}$ combinaciones distintas y necesitamos 20 bits para codificarlas.

Supongamos que hay 256 combinaciones de cuatro letras que son más frecuentes. Estas 256 combinaciones las incluimos en un diccionario y codificamos de la forma siguiente:

- si la palabra es del diccionario mandamos un 1 seguido del índice de 8-bits,
- si no está en el diccionario mandamos un 0 seguido de la representación en 20 bits.

¿Cuál es el número medio de bits por símbolo enviado? (palabra del diccionario o palabra no incluida en él)

$$R=9p+21(1-p)=21-12p$$

donde p es la probabilidad de que aparezca alguno de los 256 patrones del diccionario. ¿Cuándo es mejor usar un diccionario?

$$21-12p<20 \quad \text{sii} \quad p>1/12\approx 0.083$$

Nuestro objetivo será mejorar lo más posible los codificadores que hemos visto en los temas anteriores para lo cual debemos esperar que p sea suficientemente grande.

Para generar un buen diccionario debemos tener una buena información sobre la fuente.

Si esta información la tenemos previa a la codificación aparecerán los llamados diccionarios estáticos. Por el contrario, si hemos de ir generando el diccionario, tendremos los diccionarios adaptativos.

Tanto los diccionarios estáticos como los adaptativos van a ser estudiados en este tema.

III. Diccionarios estáticos

La elección de un diccionario estático es conveniente cuando se tiene a priori mucha información sobre la fuente.

Consideremos, por ejemplo, los datos correspondientes a un estudiante en una universidad (seguiremos el modelo americano como Sayood en [1]).
El diccionario contendrá seguro: “Name”, “Student ID” y muy probablemente “Sophomore”, “credits”. Dependiendo de la universidad algunos números puedes ser muy frecuentes. Por ejemplo, en Nebraska la mayoría de los ID de los estudiantes comienzan con 505. Estas ideas conducen a una codificación por diccionario estático.

Un modelo de diccionario estático que no es tan específico como el anterior es el que describiremos a continuación.

III.1 Codificación Digrama

Una de las formas más usuales de diccionario estático es la codificación digrama.

El diccionario está formado por todos los símbolos del alfabeto fuente más todos los pares de símbolos, llamados digramas, que pueden incluirse en un diccionario de tamaño dado.

Por ejemplo, si queremos construir un diccionario de tamaño 256 para codificación digrama de todos los caracteres ASCII imprimibles tendríamos:

- 95 entradas para los caracteres ASCII imprimibles,
- más 161 entradas para los pares de caracteres utilizados más frecuentemente.

La codificación sería la siguiente:

1. Leer dos caracteres,
2. Buscar en el diccionario para ver si el par de caracteres está incluido en él.
3. Si está, transmitimos el código correspondiente al par.
4. Si no, codificar el primer carácter y el segundo carácter se convierte en el primero del siguiente digrama.

Ejemplo 5.3.1

La fuente consiste en el alfabeto A={a,b,c,d,r} y disponemos de tres bits para la codificación.
Basándonos en el conocimiento de la fuente utilizamos la tabla

Código	Entrada	Código	Entrada
000	a	100	r
001	b	101	ab
010	c	110	ac
011	d	111	ad

Queremos codificar

abracadabra

ab	101
abra	101100 (a queda sin codificar)
abrac	101100110
abracad	101100110111
abracadab	101100110111101
abracadabra	10110011011101100 (a queda sin codificar)
abracadabra	101100110111101100000

Código	Entrada	Código	Entrada
000	a	100	r
001	b	101	ab
010	c	110	ac
011	d	111	ad

Problemas con los diccionarios estáticos. Las tablas siguientes muestran los pares de letras más frecuentes en un programa en C y en un fichero LaTeX. Son muy diferentes.

No parece razonable usar el mismo diccionario estático para ambos casos. Necesitamos diccionarios que se adapten a los datos a comprimir.

Pareja	Veces	Pareja	Veces	Pareja	Veces
e␣	1128	en	392	d␣	272
␣t	838	on	385	␣o	266
␣␣	823	n␣	353	io	257
th	817	ti	322	co	256
he	712	␣i	317	re	247
in	512	ar	314	␣$	246
s␣	494	at	313	r␣	239
er	433	␣w	309	di	230
␣a	425	te	296	ic	229
t␣	401	␣s	295	ct	226

Los 30 pares de caracteres más frecuentes en un documento LaTeX de 41364 caracteres. ␣ significa espacio en blanco

Pareja	Veces	Pareja	Veces	Pareja	Veces
b̲b̲	5728	,b̲	554	or	374
nlb̲	1471	nlnl	506	rb̲	373
;nl	1133	b̲f	505	en	371
in	985	eb̲	500	er	358
nt	739	b*	317	ri	357
=b̲	687	st	442	at	352
b̲i	662	le	440	pr	351
tb̲	615	ut	440	te	349
b̲=	612	f(	416	an	348
);	558	ar	381	lo	347

Los 30 pares de caracteres más frecuentes en una colección de programas en C con 64983 caracteres. b̲ significa espacio en blanco y nl new line.

IV. Diccionarios Adaptativos

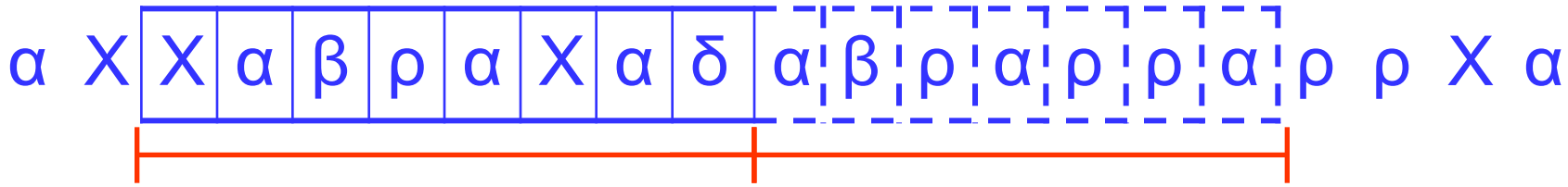
La mayoría de las técnicas de compresión basadas en diccionarios adaptativos están basadas en los trabajos de Jacob Ziv y Abraham Lempell en 1977 y 1978. Estos trabajos proporcionan dos alternativas diferentes para la construcción de diccionarios adaptativos.

Los algoritmos de compresión basados en estos trabajos reciben los nombres de LZ77 y LZ78 respectivamente, a veces también son llamados LZ1 y LZ2, respectivamente. Comenzamos estudiando el algoritmo LZ77.

- Ziv, Jacob; Lempel, Abraham (May 1977). "A Universal Algorithm for Sequential Data Compression". IEEE Transactions on Information Theory. 23 (3): 337–343.
- Ziv, Jacob; Lempel, Abraham (September 1978). "Compression of Individual Sequences via Variable-Rate Coding". IEEE Transactions on Information Theory. 24 (5): 530–536.

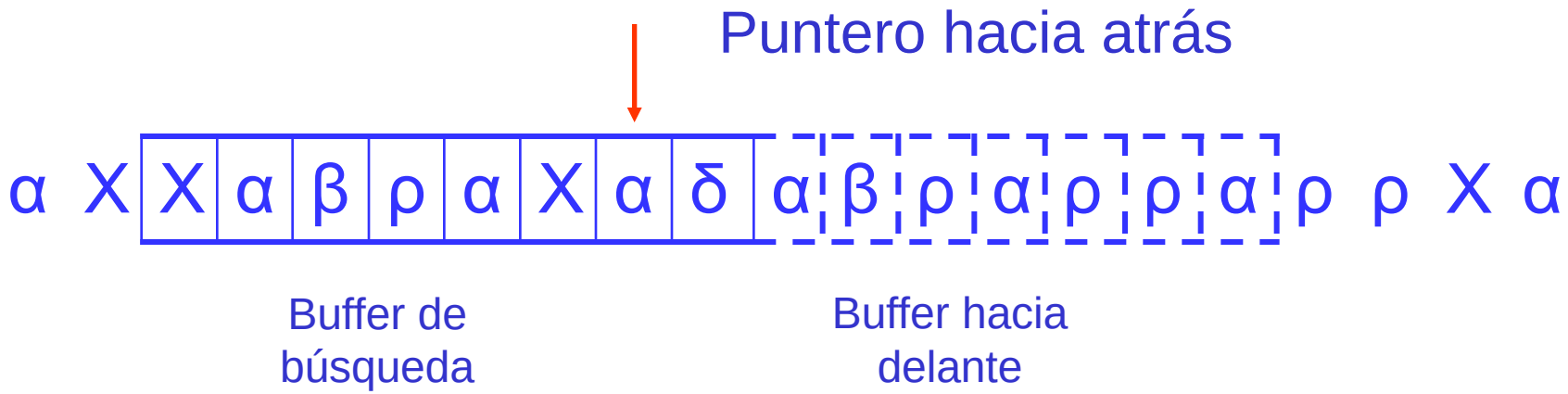
IV.1 Aproximación LZ77

En el algoritmo LZ77 el diccionario es simplemente un trozo de la secuencia previamente codificada. El codificador examina la entrada a través de una ventana que se desplaza.

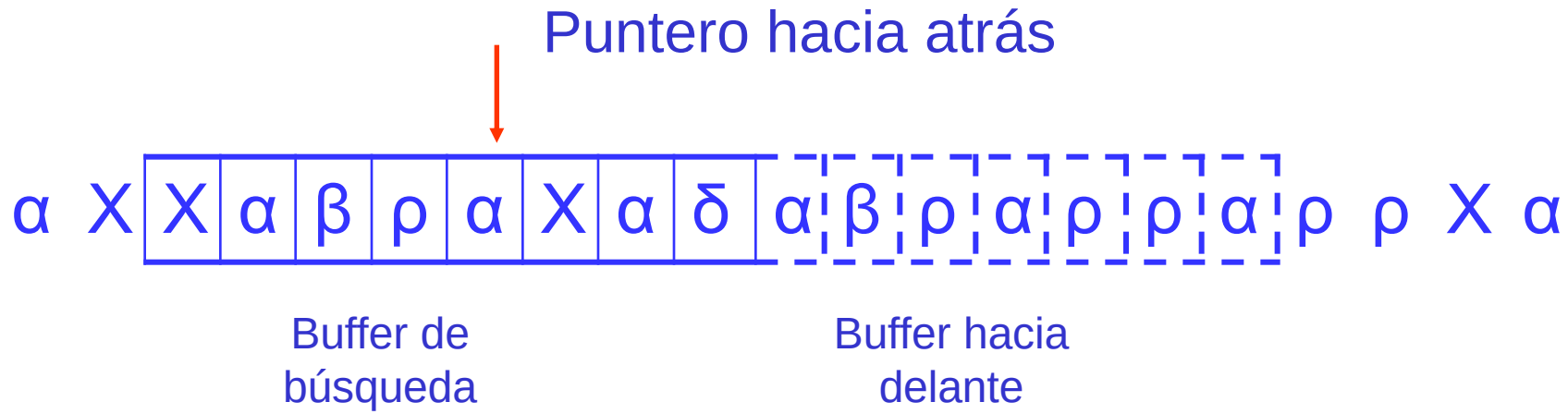


La ventana contiene un buffer de búsqueda, con un trozo de la secuencia que ha sido codificada recientemente, y el buffer hacia delante contiene el siguiente trozo de la secuencia que va a ser codificado.

Para codificar una secuencia en el buffer hacia delante el algoritmo mueve un puntero hacia atrás hasta que encuentra un símbolo igual al que queremos codificar



1. La distancia entre el puntero y el comienzo del buffer hacia delante recibe el nombre de desplazamiento (offset).
2. Miramos si los símbolos que siguen el puntero coinciden con símbolos consecutivos del buffer hacia delante. En nuestro caso δ no coincide con β.
3. Buscamos otros símbolos α en el buffer de búsqueda.



En este nuevo caso sólo han coincidido en un símbolo.

Examinamos más casos

En este nuevo caso la coincidencia es en cuatro símbolos.

Una vez encontrada la secuencia más larga que coincide, el codificador codifica el trozo de la secuencia enviando una terna (o,l,c) donde:

1. o es el offset (desplazamiento), 7 en nuestro ejemplo.
2. l es la longitud de la secuencia que coincide, 4 en nuestro ejemplo, y
3. c es el código que corresponde al siguiente carácter en el buffer hacia delante, ρ en nuestro caso.

Para codificar un símbolo que no está en el buffer hacia atrás enviamos su código. En este caso la terna sería (0,0,código del símbolo).

Supongamos que el tamaño del buffer de búsqueda es S, que W es el tamaño del buffer hacia delante y A es el tamaño del alfabeto fuente. Entonces, cada terna necesita un número de bits N donde

$$N = \lceil \log_2 S \rceil + \lceil \log_2 W \rceil + \lceil \log_2 A \rceil$$

Observa que el segundo término de la suma depende de W y no de S, en el ejemplo siguiente estudiaremos la razón.

Veamos los tres casos con los que nos podemos encontrar cuando vamos a codificar:

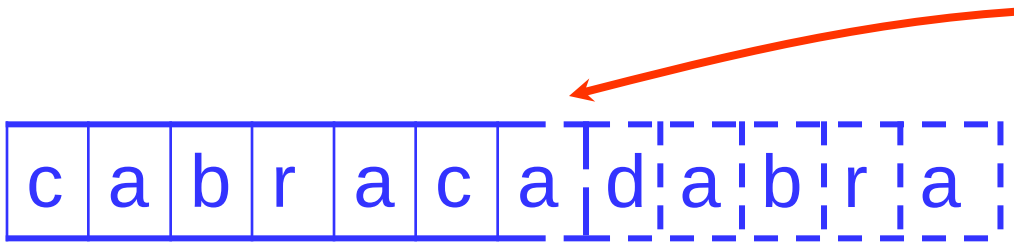
1. No hay un carácter igual al que queremos codificar en el buffer de búsqueda
2. Sí hay,
3. La hilera de caracteres encontrada incluye letras del alfabeto que están en el buffer hacia delante.

Ejemplo 5.4.1: Aproximación LZ77.

Queremos codificar la secuencia

cabracadabrarrarrad

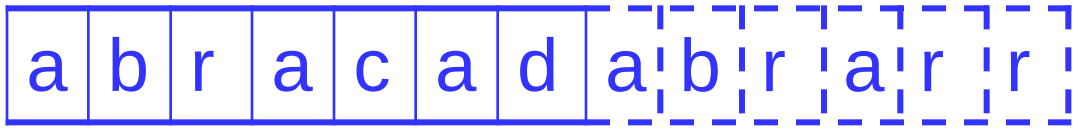
La longitud de la ventana es 13, el buffer hacia delante tiene tamaño 6 y nos encontramos en la siguiente situación de "búfferes" (piensa como hemos codificado cabrac)



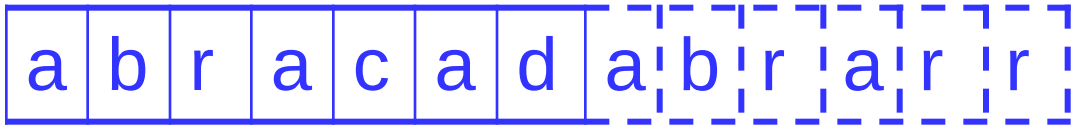
Para codificar la a utilizamos (2,1,C(d))

Parece un gasto algo inútil

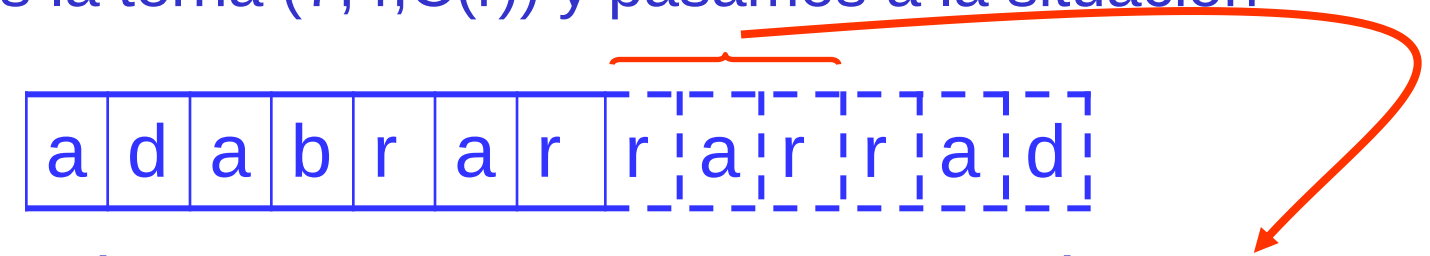
Una vez codificado d nos queda



Para la configuración (repetimos final de la transparencia anterior)



codificamos la terna (7,4,C(r)) y pasamos a la situación



Ahora pensaríamos que la siguiente terna sería (3,3,C(r)) pero NO.

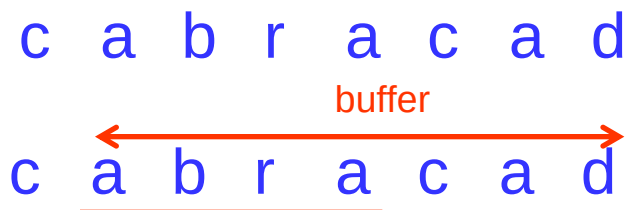
La codificación será (3,5,C(d)) lo explicaremos cuando ahora decodifiquemos.

Observa: las tres últimas letras del buffer de búsqueda (rar) coinciden con las tres primeras del buffer hacia delante, pero además las 2 letras siguientes del buffer hacia delante (ra antes de d) coinciden con las dos primeras del buffer hacia delante.

Veamos ahora la decodificación.

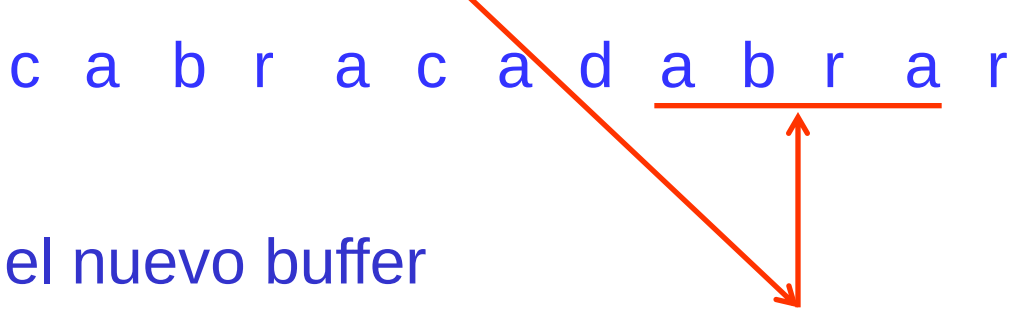
Supongamos que ya hemos decodificado cabrac

(2,1,C(d)) produce

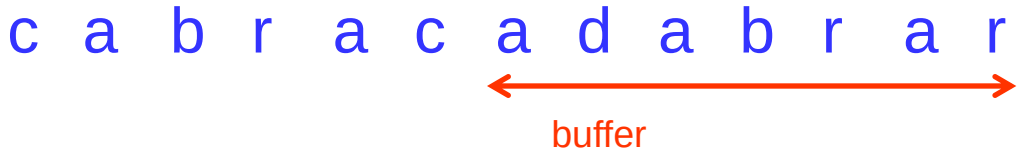


El buffer queda

(7,4,C(r)) produce



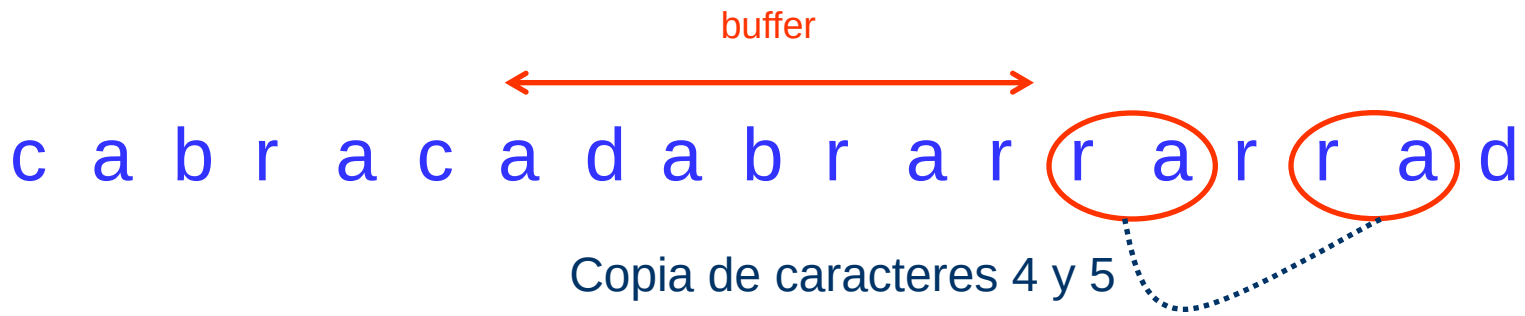
y tenemos la decodificación y el nuevo buffer



Nos llega ahora la terna (3,5,C(d)).

- Los tres primeros elementos que añadimos son rar
- Tenemos rar (fin de buffer) rar
- Puedo coger dos más después del buffer (ra)
- Es decir tengo (añadiendo C(d))

(3,5,C(d)



Observa que si los caracteres finales fuesen, por ejemplo, rarrard en lugar de rarrad su codificación sería (3,6,C(d)).

¿y si los caracteres fuesen rarrarrard?. Recuerda que no podemos salirnos del buffer hacia delante.

Observa que el uso de la ventana supone que los patrones que se repiten están cerca unos de otros, esto será modificado en el algoritmo LZ78. Antes veamos algunas extensiones de LZ77.

Variaciones del método LZ77

1. Para mejorar la codificación de las ternas podemos usar, en lugar de una codificación de longitud fija, una codificación de longitud variable. Métodos como **PKZip, Zip**, **Lharc**, **PNG**, **gzip** y **Arj** usan LZ77 seguido de codificación de longitud variable.
2. Variaciones de LZ77 también incluyen la modificación de los tamaños de buffer y la búsqueda eficiente en ellos.
3. Por último, en lugar de codificar la terna podemos poner antes una etiqueta para saber si viene un carácter único (por ejemplo, (0,0,C(d)) o la codificación de varios y así distinguir los dos casos. En el segundo sólo mandamos el desplazamiento y la longitud y en el primero el código del carácter que aparece solo. Esta variante de LZ77 la implementa LZSS que es utilizado en **Rar**.

IV.2 Aproximación LZ78

LZ77 supone que los patrones similares aparecen más o menos juntos. Sin embargo, si una secuencia vuelve a aparecer después de un tiempo superior al tamaño del buffer de búsqueda, surgen problemas. Consideremos la secuencia

a b c d e f a b c d e f a b c d e f

Búffer de búsqueda Búffer hacia delante

con los búferes de búsqueda y búsqueda hacia delante marcados. Observa:

- Sólo vamos a poder codificar ternas de la forma (0,0,C(f)), (0,0,C(a)),… o si usamos LZSS codificaríamos cada carácter precedido por la correspondiente bandera. En resumen, no comprimimos, al revés, expandimos.
- Observa, también, que con un carácter más en el buffer de búsqueda todo sería muy diferente.

Aunque el ejemplo anterior es extremo, hay circunstancias, no tan extremas, en las que el tamaño finito del buffer de búsqueda constituye un problema.

El algoritmo LZ78 resuelve este problema creando un verdadero diccionario.

La construcción del diccionario se tiene que hacer en el codificador y decodificador de la misma forma.

Las entradas son codificadas como pares (i,c)

- i es el índice en el diccionario cuyo contenido coincide con la mayor hilera (la más larga) de letras del alfabeto a codificar y
- c es el código del siguiente carácter (el primer carácter después de la hilera).

Un índice 0 se utiliza para notar que el carácter a codificar no tiene igual en el diccionario.

Cada nuevo elemento del diccionario se construye concatenado los símbolos que corresponden al índice i y el símbolo c. Veamos un ejemplo.

Supongamos que la secuencia a codificar es

wabbaḇwabbaḇwabbaḇwabbaḇwooḇwooḇwoo

El diccionario está vacío por lo que los primeros símbolos serán codificados con un par que comienza por cero. Las tres primeras codificaciones son

(0,C(w)) (0,C(a)) (0,C(b))

que produce el diccionario

Índice	Entrada
1	w
2	a
3	b

wabbabwabbabwabbabwabbabwoobwoobwoo

El cuarto símbolo es b que ya está en el diccionario pero ba no está.

Enviamos entonces (3,C(a)). Nuestro diccionario pasa a ser

Índice	Entrada
1	w
2	a
3	b
4	ba

wabbab̲wabbab̲wabbab̲wabbab̲woob̲woob̲woo

ya teníamos

Salida del codificador	Diccionario	
	Indice	Entrada
(0,C(w))	1	w
(0,C(a))	2	a
(0,C(b))	3	b
(3,C(a))	4	ba

codificación desde hasta

Salida del codificador	Diccionario	
	Indice	Entrada
(6,C(b))	9	wab
(4,C(b̲))	10	bab̲
(9,C(b))	11	wabb
(8,C(w))	12	ab̲w

codificación desde hasta

Salida del codificador	Diccionario	
	Indice	Entrada
(0,C(b̲))	5	b̲
(1,C(a))	6	wa
(3,C(b))	7	bb
(2,C(b̲))	8	ab̲

codificación desde hasta el final

Salida del codificador	Diccionario	
	Indice	Entrada
(0,C(o))	13	o
(13,C(b̲))	14	ob̲
(1,C(o))	15	wo
(14,C(w))	16	ob̲w
(13,C(o))	17	oo

Para la decodificación

- Tendríamos la secuencia

(0,C(w)) (0,C(a)) (0,C(b)) (3,C(a)) (0,C(b̲)) (1,C(a)) (3,C(b)) (2,C(b̲)) (6,C(b)) (4,C(b̲)) (9,C(b)) (8,C(w)) (0,C(o)) (13,C(b̲)) (1,C(o)) (14,C(w)) (13,C(o))

- Junto con los códigos de los símbolos.

¿Sabrías decodificarla?

Recuerda que tienes que ir construyendo el diccionario al mismo tiempo.

IV.3. Codificación Lempel-Ziv-Welsh (LZW)

Hay diferentes formas de formas de modificar el algoritmo LZ78.

Sin lugar a duda la más conocida es la modificación debida a Welch que es conocida como LZW.

Su idea básica es suprimir la necesidad de codificar el segundo elemento del código, es decir, el código del símbolo final.

IV.3. Codificación Lempel-Ziv-Welsh (LZW)

El algoritmo LZW funciona como sigue:

1. El diccionario (conocido por el codificador y el decodificador) comienza con todas las letras del alfabeto.

2. El input *p* al codificador va creciendo con nuevos caracteres *a* mientras *pa* esté en el diccionario.

3. Si pa no está en el diccionario transmitimos el índice de *p* y el patrón *pa* se añade al diccionario y comenzamos con el nuevo input que comienza por *a*.

Vamos a ver estas ideas con un ejemplo en el codificador y en el decodificador. Pero antes incluyamos el algoritmo

Algoritmo Codificación Lempel-Ziv-Welsh (LZW)

- s= get input character
- While there are still input characters do
    c = get input character
    If s+c is in the dictionary
        s=s+c
    Else
        output the code for s
        add s+c to the dictionary with a new code
        s = c
    End of if
- End of While
- output the code for s

Ejemplo 6.1: Codificación en el algoritmo LZW

Supongamos que la secuencia a codificar es

ABABBABCABABBA

y que hemos utilizado del 1 al 3 para codificar los símbolos unitarios

s	c	output	code	string
			1	A
			2	B
			3	C

Ejemplo 6.1: Codificación en el algoritmo LZW

Supongamos que la secuencia a codificar es

ABABBABCABABBA

y que hemos utilizado del 1 al 3 para codificar los símbolos unitarios

s	c	output	code	string
			1	A
			2	B
			3	C
A	B	1	4	AB
B	A	2	5	BA
A	B			
AB	B	4	6	ABB
B	A			
BA	B	5	7	BAB
B	C	2	8	BC
C	A	3	9	CA

s	c	output	code	string
A	B			
AB	A	4	10	ABA
A	B			
AB	B			
ABB	A	6	11	ABBA
A		1		

Salida:

1 2 4 5 2 3 4 6 1

Ejemplo 6.2: Codificación en el algoritmo LZW

Supongamos que la secuencia a codificar es

/WED/WE/WEE/WEB/WET

y que hemos utilizado 0-255 para codificar todos los símbolos unitarios.

La tabla inicial es

s	c	output	code	string
			47	/
			66	B
			68	E
			69	D
			84	T
			87	W

Ejemplo 6.2: Codificación en el algoritmo LZW

/WED/WE/WEE/WEB/WET

s	c	output	code	string
			47	/
			66	B
			68	E
			69	D
			84	T
			87	W
/	W	47	256	/W
W	E	87	257	WE
E	D	68	258	ED
D	/	69	259	D/
/	W			
/W	E	256	260	/WE

s	c	output	code	string
E	/	68	261	E/
/	W			
/W	E			
/WE	E	260	262	/WEE
E	/			
E/	W	261	263	E/W
W	E			
WE	B	257	264	WEB
B	/	66	265	B/
/	W			
/W	E			
/WE	T	260	266	/WET
T		84		

Salida: 47 87 68 69 256 68 260 261 257 66 260 84

Algoritmo de decodificación Lempel-Ziv-Welsh (LZW) (un poco demasiado simple)

```
s=NIL; %hilera vacía
WHILE not eof DO
        k= next input code
        entry = dictionary entry for k
        output entry
        if (s!=NIL)
                c = first character in entry
                add s + c to the dictionary
        end if
        s = entry
END of WHILE
```

Ejemplo 6.1: Decodificación en el algoritmo LZW

Supongamos que la secuencia a decodificar es

1 2 4 5 2 3 4 6 1

y que hemos utilizado del 1 al 3 para codificar los símbolos unitarios

s	k	Output/entry	code	string
			1	A
			2	B
			3	C
Nil	1	A		
A	2	B	4	AB
B	4	AB	5	BA
AB	5	BA	6	ABB
BA	2	B	7	BAB
B	3	C	8	BC
C	4	AB	9	CA
AB	6	ABB	10	ABA

s	c	output	code	string
ABB	1	A	11	ABBA

Original
ABABBABCABABBA

Salida
ABABBABCABABBA

Ejemplo 6.2: Decodificación en el algoritmo LZW

Supongamos que la secuencia a decodificar es

47 87 68 69 256 68 260 261 257 66 260 84

hemos utilizado 0-255 para codificar todos los símbolos unitarios

s	k	output/entry	code	string
			47	/
			66	B
			68	E
			69	D
			84	T
			87	W
NIL	47	/		
/	87	W	256	/W
W	68	E	257	WE
E	69	D	258	ED
D	256	/W	259	D/
/W	68	E	260	/WE

s	k	output/entry	code	string
E	260	/WE	261	E/
/WE	261	E/	262	/WEE
E/	257	WE	263	E/W
WE	66	B	264	WEB
B	260	/WE	265	B/
/WE	84	T	266	/WET

Original
/WED/WE/WEE/WEB/WET

Salida
/WED/WE/WEE/WEB/WET

Problemas y soluciones para el algoritmo LZW

Secuencia a codificar: a b a b a b a b

s	c	output	code	string
			1	a
			2	b
a	b	1	3	ab
b	a	2	4	ba
a	b			
ab	a	3	5	aba
a	b			
ab	a			
aba	b	5	6	abab
b		2		

Decodificar 1 2 3 5 2

s	k	output/entry	code	string
			1	a
			2	b
NIL	1	a		
a	2	b	3	ab
b	3	ab	4	ba
ab	5	???		

Código: 1 2 3 5 2

- La entrada 5 no está todavía en el decodificador ???.
- La entrada 5 debe estar formada por ab y la primera letra que sólo puede ser a ya que si no la entrada 5 (QUE ES LO SIGUIENTE A DECODIFICAR) no empezaría por a.
- ¿Cómo lo arreglamos?.

Problemas y soluciones para el algoritmo LZW

Otro ejemplo. Supongamos que el alfabeto es {A,B,C} y que la secuencia a codificar es ABABBABCABBABBA…

s	c	Output	code	string
			1	A
			2	B
			3	C
A	B	1	4	AB
B	A	2	5	BA
A	B			
AB	B	4	6	ABB
B	A			
BA	B	5	7	BAB
B	C	2	8	BC
C	A	3	9	CA
A	B			
AB	B			
ABB	A	6	10	ABBA
A	B			
AB	B			
ABB	A			
ABBA		10		

Decodifiquemos la secuencia
1 2 4 5 2 3 6 10 …

s	k	output/entry	code	string
			1	A
			2	B
			3	C
NIL	1	A		
A	2	B	4	AB
B	4	AB	5	BA
AB	5	BA	6	ABB
BA	2	B	7	BAB
B	3	C	8	BC
C	6	ABB	9	CA
ABB	10	???		

La entrada 10 no está todavía en el decodificador ???.

No obstante la entrada 10 debe estar formada por ABB y la primera letra que sólo puede ser A ya que si no la entrada 10 no empezaría por A.

¿Cómo lo arreglamos?.

Problemas y soluciones para el algoritmo LZW

¿Cuándo ocurre este problema?

En nuestro diccionario tenemos ya una secuencia de la forma

carácter+string

Al codificador llega en algún momento (por primera vez) la secuencia

carácter+string+carácter

No tiene ningún problema, manda el código de

carácter+string

y crea una entrada en el diccionario para

carácter+string+carácter

Problemas y soluciones para el algoritmo LZW

¿Qué ocurre si en el codificador a la secuencia

carácter+string+carácter

le sigue

carácter+string

?

El codificador no tiene problema, manda el código de

carácter+string+carácter

Desgraciadamente, el decodificador no ha sido capaz de verlo todavía. Este nuevo código que todavía no ha visto no puede ser otro que el correspondiente a

carácter+string+carácter

Algoritmo de decodificación Lempel-Ziv-Welsh (LZW) (sin errores)

```
s=NIL; %hilera vacía
WHILE there are still input characters DO
        k= next input code
        entry = dictionary entry for k
        /* manejo de excepción */
        If (entry==NULL)
                entry=s+first character in s
        output entry
        if (s!=NIL)
                c = first character in entry
                add s + c to the dictionary
        end if
        s = entry
END of WHILE
```

Problemas y soluciones para el algoritmo LZW

Decodifiquemos bien la secuencia 1 2 3 5 2

s	k	output/entry	code	string
			1	a
			2	b
NIL	1	a		
a	2	b	3	ab
b	3	ab	4	ba
ab	5	aba	5	aba
aba	2	b	6	abab

Original a b a b a b a b

Decodificada a b a b a b a b

Problemas y soluciones para el algoritmo LZW

Decodifiquemos bien ahora el segundo ejemplo con problemas
1 2 4 5 2 3 6 10 …

s	k	output/entry	code	string
			1	A
			2	B
			3	C
NIL	1	A		
A	2	B	4	AB
B	4	AB	5	BA
AB	5	BA	6	ABB
BA	2	B	7	BAB
B	3	C	8	BC
C	6	ABB	9	CA
ABB	10	ABBA	10	ABBA

Original:

ABABBABCABBABBA…

Decodificada:

ABABBABCABBABBA…

Cualquier implementación razonable requiere que exista alguna limitación para el tamaño del diccionario:

- Por ejemplo en GIF se utilizan 4096 entradas, es decir, 12 bits, que es más largo que los 8 bits originales del ASCII.
- En implementaciones reales la longitud del código se mantiene en [l0,lmax]. Por ejemplo, para compress en Unix l0=9 y lmax=16 por defecto.
- Se empieza 2^{l0} y cuando se llena se aumenta el código con un bit así hasta llegar a 2^{lmax}.

Observa que si no hay estructura repetitiva en los datos, el tamaño del fichero comprimido puede ser mayor que el original. El sistema podría cambiar a un modo sin compresión y usar 8 bits cuando esto ocurre.

¿Qué ocurre cuando superamos el tamaño del diccionario?

- Compress inicializa el diccionario
- También podríamos eliminar las entradas que menos se han utilizado recientemente. Para ello basta que busquemos que entradas no son prefijo de otras

V. Implementaciones reales que usan LZW

Compress

Fue el estándar de compresión de UN*X hasta hace algunos años. Usa una modificación de LZW llamada LZC que infringía la patente sobre LZW de Unisys e IBM.

Para obtener una mejor compresión, cuando el diccionario tiene menos de 256 cadenas usa 9 bits para representar cada una de ellas, cuando menos de 512 usa 10 bits y así continúa hasta que se alcanza el límite impuesto por la opción -b (por defecto es 16).

El valor de -b puede modificarse en el código fuente para acomodarse a máquinas con poca capacidad.

Cuando se llega al límite de bits (valor con –b), compress comprueba periódicamente la razón de compresión.

- Si se incrementa continúa usando el mismo diccionario.
- Si decrece, compress descarta todo el diccionario y lo construye de nuevo.

De esta forma se adapta al nuevo "bloque" en el fichero.

GIF

Graphics Interchange Formato (GIF) es un formato de almacenamiento y compresión de imágenes. Fue ideado originalmente por UNISYS y CompuServe para transmitir imágenes por teléfono a través de modem.

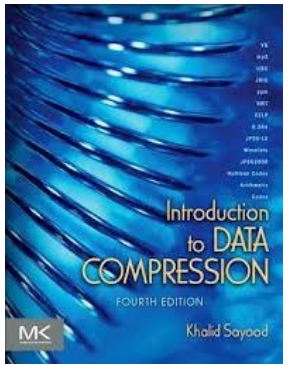
GIF puede representar imágenes con una paleta de 2 a 256 colores que se codifican usando el algoritmo LZW.

Toda la información sobre el formato puede encontrarse en http://tronche.com/computer-graphics/gif/

También utilizan LZW **pdf, compress, gzip, winzip** y **WinRar**

VI Bibliografía

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



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