



Universidad del  
**Rosario**

---

**Teoría de la información**

---

Realizado por  
Esteban Hernández Ramírez  
CC: 1'007.658.073

Supervisado por  
PhD. Carlos Eduardo Álvarez Cabrera

En el curso  
Opción de grado 1

Trabajo presentado a  
Comité de evaluación de opción de grado

Escuela de Ingeniería, Ciencia y Tecnología  
Matemáticas Aplicadas y Ciencias de la Computación  
Universidad del Rosario  
April 30, 2022

Bogotá-Colombia

## Contents

<b>1</b>	<b>Nuevos objetivos</b>	<b>2</b>
1.1	Objetivo general . . . . .	2
1.2	Objetivos específicos . . . . .	2
<b>2</b>	<b>Introducción</b>	<b>2</b>
<b>3</b>	<b>La relación entre la entropía relativa y la compresión de cadenas</b>	<b>3</b>
3.1	La entropía relativa como una medida de comparación entre cadenas . . . . .	3
3.2	Algoritmo de compresión LZ77 . . . . .	5
3.2.1	Algoritmo de compresión sin pérdida . . . . .	5
3.2.2	Codificación de diccionario . . . . .	5
3.2.3	Codificación de diccionario estático . . . . .	5
3.2.4	Codificación de diccionario semi-adaptativo . . . . .	5
3.2.5	Codificación de diccionario adaptativo . . . . .	5
3.2.6	Extensión reproducible de una cadena . . . . .	5
3.2.7	Ejemplo de codificación LZ77 . . . . .	8
3.2.8	Versión funcional . . . . .	9
3.3	Calculando la Entropía Relativa por medio de la compresión de archivos . . . . .	10
3.3.1	Definiendo el criterio de desempeño . . . . .	10
3.3.2	Acotando la tasa de compresión máxima alcanzable . . . . .	11

# 1 Nuevos objetivos

Debido a la amplitud del tema se consideró reducir el objetivo de hacer una revisión bibliográfica en general del concepto de información a una aplicación particular de la teoría de la información en la compresión de cadenas.

## 1.1 Objetivo general

Revisar en la bibliografía de qué manera se aplica la teoría de la información a la compresión de cadenas.

## 1.2 Objetivos específicos

1. Revisar algunas de las definiciones de información más estudiadas en la literatura.
2. Revisar en la literatura de qué manera estas definiciones de información se aplican a la compresión de cadenas.
3. Implementar el algoritmo de Lempel-Ziv para la compresión de cadenas.
4. Observar los efectos del algoritmo de Lempel-ziv sobre una cadena en términos de las medidas de información.
5. Revisar una aplicación del algoritmo de Lempel-ziv para clasificar lenguajes.

# 2 Introducción

La idea de una máquina de Turing que retorna una secuencia infinita de 1's y 0's, luego de haber operado sobre ella bajo reglas codificadas en otra secuencia de 1's y 0's, nos lleva a la pregunta de si dada una secuencia cualquiera, es posible determinar a partir de ella la secuencia de reglas codificadas que la generaron: más importante aún, cómo determinar la secuencia de reglas codificadas más corta capaz de generar la secuencia dada. De hecho, este problema ha sido formulado rigurosamente en la teoría de la computación como la Complejidad de Kolmogórov; introducida independientemente en la década de los 60's por Solomonoff ([8]), Kolmogorov ([4]), y Chaitin ([2]).

Se sabe que si la forma más sucinta de expresar la cadena fuera ella misma, entonces la cadena habrá sido generada de manera completamente aleatoria ([6]). También se sabe que, de manera general, la tarea de determinar la secuencia de reglas codificadas más corta es incomputable ([10]). No obstante, los esfuerzos de la comunidad científica se han centrado últimamente en idear formas de comprimir cualquier secuencia, lo más que sea posible. El trabajo pionero de Lempel y Ziv ([11]) demostró que existen formatos de compresión para los cuales el tamaño óptimo de un archivo comprimido tiende a estimar la entropía de la fuente generadora. Esta "entropía" fué una medida propuesta por Claude Shannon en la década de los 50's ([7]): esta medida es continua y es máxima cuando la probabilidad de generar cualquier símbolo en la secuencia es la misma. Lo anterior ha permitido calcular una aproximación a la Complejidad de Kolmogorov de una secuencia. Otras aproximaciones como [9] confirmarían más adelante lo visto por Lempel y Ziv, y acabaría por reforzar la creencia de que la Complejidad de Kolmogórov y la Entropía de Shannon están más relacionadas de lo que podría parecer en un principio.

Lo anterior posibilita diferenciar las reglas (aún sin conocerlas del todo) que generan algunas secuencias, usando los patrones que puedan presentarse en las propias secuencias para comprimirlas. Esto se ejemplifica

perfectamente para el caso de textos (secuencias) escritos en algún lenguaje natural en diferentes idiomas: midiendo la entropía de sus fuentes mediante la compresión de las cadenas se puede distinguir un lenguaje del otro. La idea intuitiva es la siguiente: en este contexto, la entropía relativa se interpreta como el número de caracteres desperdiciados para codificar una cadena generada por una fuente  $\mathcal{B}$  mediante la codificación óptima para las cadenas generadas por otra fuente  $\mathcal{A}$  [1]. No en vano, para aproximar la entropía se pueden emplear algoritmos de compresión como lo demostraron Lempel y Ziv en ([11]). Y en ese orden de ideas, la entropía relativa por caracter entre las fuentes  $\mathcal{A}$  y  $\mathcal{B}$ ,  $S_{AB}$ , puede aproximarse como:

$$S_{AB} = \frac{\Delta_{Ab} - \Delta_{Bb}}{|b|}, \quad (1)$$

tal cómo se definió en [1]: Siendo  $\Delta_{AB} = L_{A+B} - L_A$  con  $L_X$  la longitud en bits del archivo comprimido  $X$ .

En realidad, es estos casos lo que nos interesará es estimar la entropía relativa entre las dos fuentes por medio de algoritmos de compresión, que es lo que describíamos intuitivamente en el párrafo anterior: ya que la entropía relativa se puede utilizar como una medida de distancia entre las cadenas generadas por dos fuentes distintas  $\mathcal{A}$  y  $\mathcal{B}$ . Es decir, qué tan bien funcionan las reglas que emplea la fuente  $\mathcal{A}$  para generar mensajes propios de la fuente  $\mathcal{B}$ , lo que nos dirá qué tan distintos son los mensajes que genera cada una.

En ese orden de ideas, en el presente escrito nos enfocamos en estudiar el problema de estimar la entropía relativa de dos fuentes generadoras de secuencias, al comprimir textos escritos en diferentes idiomas y detectar los cambios en el tamaño de los textos comprimidos. La forma en la que se organiza el texto a continuación es la siguiente: 1) cómo calcular la entropía relativa para textos escritos en lenguaje natural. 2) cómo detectar los cambios entre una secuencia y otra. 3) cómo interpretar estos cambios en términos de distancias filogenéticas del lenguaje. 4) Hacer una revisión de la aplicación de estos principios al análisis del ADN. 5) Implementación del algoritmo para buscar distancias entre algunas secuencias de ADN.

### 3 La relación entre la entropía relativa y la compresión de cadenas

En esta sección, explicamos la relación que existe entre la entropía relativa y la codificación de cadenas. Específicamente en cómo el problema inicial se reduce a un tipo particular de esto último: la compresión de cadenas.

#### 3.1 La entropía relativa como una medida de comparación entre cadenas

Para saber qué tan bien funcionan las reglas que emplea una fuente  $\mathcal{A}$  para generar mensajes propios de otra fuente  $\mathcal{B}$ , lo que nos dirá qué tan distintos son los mensajes que genera cada una, se puede utilizar el número de caracteres desperdiciados para codificar una secuencia emitida por la fuente  $\mathcal{B}$  con el código óptimo para la fuente  $\mathcal{A}$ : la entropía relativa entre las fuentes  $\mathcal{A}$  y  $\mathcal{B}$  (Revisar Kullback-Leibler [13]).

Lo anterior se justifica de la siguiente manera, suponga que se tienen dos fuentes ergódicas  $\mathcal{A}$  y  $\mathcal{B}$  que están emitiendo secuencias de '0' y '1':  $\mathcal{A}$  emitiendo '0' con probabilidad  $p$  y '1' con probabilidad  $1 - p$ , mientras que  $\mathcal{B}$  emite un '0' con probabilidad  $q$  y un '1' con probabilidad  $1 - q$ . La codificación óptima de las secuencias de la fuente  $\mathcal{A}$  consiste en emplear  $\log_2\left(\frac{1}{p}\right)$  bits para codificar el '0' y  $\log_2\left(\frac{1}{1-p}\right)$  para codificar el '1', mientras que para la fuente  $\mathcal{B}$ , la codificación óptima emplea  $\log_2\left(\frac{1}{q}\right)$  bits para codificar el '0' y  $\log_2\left(\frac{1}{1-q}\right)$  para codificar el '1'. De esta manera se estará empleando la menor cantidad de bits posible para codificar los mensajes de ambas fuentes: llámese la entropía de Shannon, tal como se definió en [7].

En ese orden de ideas, la entropía por caracter de una secuencia emitida por  $\mathcal{B}$  en términos de la codificación óptima para  $\mathcal{A}$  puede obtenerse a partir de una secuencia infinitamente larga de la fuente  $\mathcal{B}$  que ha

sido codificada con las cantidades de bits de la fuente  $\mathcal{A}$ . Para esto, expresamos la cantidad de símbolos en la codificación de la secuencia como la suma de la cantidad de unos y ceros que aparecen en la secuencia codificada. Luego dividimos la suma de la cantidad de unos y ceros entre la longitud de la cadena: la idea es que estamos considerando que  $\{X_i\}_{i=0}^{\infty}$  es una muestra aleatoria infinitamente grande de variables aleatorias independientes de Bernoulli con media  $\mu = q$ , entonces

$$\lim_{N \rightarrow \infty} \frac{1}{N} \cdot \left[ \log_2 \left( \frac{1}{p} \right) \sum_{i=0}^N X_i + \log_2 \left( \frac{1}{1-p} \right) \left( N - \sum_{i=0}^N X_i \right) \right] = q \cdot \log_2 \left( \frac{1}{p} \right) + (1-q) \cdot \log_2 \left( \frac{1}{1-p} \right)$$

por el teorema del limite central. Éste último afirma que el estimador estadístico de la media muestral,

$$\frac{1}{N} \sum_{i=0}^N X_i,$$

converge al parámetro  $p$  de la distribución de Bernoulli de las variables aleatorias de su muestra, conforme  $N$ , el tamaño de la muestra, tiende a infinito.

Sin embargo, para que esto último sea cierto, necesitamos que la muestra esté constituida por variables aleatorias independientes e idénticamente distribuidas. No obstante, estas hipótesis de independencia para el teorema están garantizadas, bajo la asunción de que  $\mathcal{A}$  y  $\mathcal{B}$  son fuentes ergódicas. Por su parte, la entropía por caracter de una secuencia emitida por  $\mathcal{B}$  en la codificación óptima para  $\mathcal{B}$  es  $q \cdot \log_2 \left( \frac{1}{q} \right) + (1-q) \log_2 \left( \frac{1}{1-q} \right)$ . Análogamente para una secuencia emitida por  $\mathcal{A}$ .

De esta manera, el número de bits por caracter desperdiciados para codificar una secuencia emitida por  $\mathcal{B}$  con la codificación óptima para  $\mathcal{A}$  se puede calcular como la diferencia:

$$\begin{aligned} S_{\mathcal{AB}} &= \left[ q \cdot \log_2 \left( \frac{1}{p} \right) + (1-q) \log_2 \left( \frac{1}{1-p} \right) \right] - \left[ q \cdot \log_2 \left( \frac{1}{q} \right) + (1-q) \log_2 \left( \frac{1}{1-q} \right) \right] \\ &= \left[ q \cdot \log_2 \left( \frac{1}{p} \right) - q \cdot \log_2 \left( \frac{1}{q} \right) \right] + \left[ (1-q) \log_2 \left( \frac{1}{1-p} \right) - (1-q) \log_2 \left( \frac{1}{1-q} \right) \right] \\ &= q \left[ \log_2 \left( \frac{1}{p} \right) - \log_2 \left( \frac{1}{q} \right) \right] + (1-q) \left[ \log_2 \left( \frac{1}{1-p} \right) - \log_2 \left( \frac{1}{1-q} \right) \right] \\ &= q \left[ \log_2 \left( \frac{q}{p} \right) \right] + (1-q) \left[ \log_2 \left( \frac{1-q}{1-p} \right) \right] \\ &= -q \cdot \log_2 \left( \frac{p}{q} \right) - (1-q) \cdot \log_2 \left( \frac{1-p}{1-q} \right), \end{aligned}$$

la entropía relativa de  $\mathcal{A}$  y  $\mathcal{B}$ . De manera análoga se define la entropía relativa de  $\mathcal{B}$  y  $\mathcal{A}$ :  $S_{\mathcal{BA}}$ .

En conclusión, midiendo la entropía relativa podemos saber qué tan distintos son los mensajes que generan dos fuentes. Así que, hemos reducido (y restringido) el problema original a medir la entropía relativa de dos fuentes. Pero entonces queda una pregunta: ¿cómo calcular esta entropía relativa?. Necesitamos un procedimiento que estime la entropía relativa de un par de cadenas de entrada. Si bien existen varias maneras de calcular la entropía relativa de dos cadenas [5], la que interesa a este escrito es la de calcular la entropía relativa por medio de la compresión de cadenas. Antes, conviene presentar el formato de codificación LZ77 y una revisión de los resultados formales que respaldan esta aproximación [11].

## 3.2 Algoritmo de compresión LZ77

### 3.2.1 Algoritmo de compresión sin pérdida

Un algoritmo de compresión sin pérdida, codifica una cadena como otra de menor tamaño, codificando las cadenas más frecuentes con códigos más cortos. Lo que caracteriza un algoritmo de compresión sin pérdida es que siempre es posible reconstruir completamente la cadena original a partir de la cadena codificada.

El algoritmo de codificación LZ77 es un algoritmo de compresión sin pérdida. Para la codificación de cadenas el algoritmo LZ77 emplea una codificación de diccionario.

### 3.2.2 Codificación de diccionario

La codificación de diccionario consiste en: construir un diccionario de patrones, almacenando cada patrón junto con una dirección de referencia; luego, se codifica una cadena, reemplazando todas las repeticiones de un patrón por su respectiva dirección en el diccionario. De esta manera, el código consiste en un conjunto de referencias que direccionan a un patrón en específico. Para decodificar una cadena codificada, se reemplaza de vuelta la referencia por su entrada en el diccionario.

### 3.2.3 Codificación de diccionario estático

Un diccionario puede ser estático, es decir, predefinido y fijo. Este tipo de diccionario comúnmente implementa estándares. Un ejemplo de esto es la codificación ASCII, que codifica cada símbolo del teclado como ocho bits: cada vez que se presiona una tecla el computador lo almacena como ocho bits en memoria.

### 3.2.4 Codificación de diccionario semi-adaptativo

Un diccionario también puede ser semi-adaptativo, es decir, hecho a mano, específicamente. para la cadena que se codifica. Para eso, primero se procesa la cadena para extraer los patrones y construir el diccionario. Luego, se procesa la cadena para codificarla. No obstante, para decodificar la cadena es necesario el diccionario con el cual se codificó, por lo que, además del código, también se debe transmitir el diccionario.

### 3.2.5 Codificación de diccionario adaptativo

Un diccionario adaptativo se construye al tiempo que se codifica una cadena. Como tal, es único para cada cadena, sin embargo, no es necesario transmitirlo junto con el código para su decodificación, ya que el diccionario está implícito en el propio código. De esta manera, el decodificador, operando apropiadamente sobre el código, puede deshacer la codificación para obtener nuevamente la cadena original.

El algoritmo de codificación LZ77 es un algoritmo de codificación de diccionario adaptativo.

### 3.2.6 Extensión reproducible de una cadena

La extensión reproducible de una cadena  $S$  en la posición  $j$  es la subcadena más larga que empieza en la posición  $j + 1$  y que ocurre antes de la posición  $j$ , en  $S$ . Por ejemplo, si tomamos  $j = 10$ , para la siguiente cadena:

*abbcababccabbac,*

entonces la extensión reproducible sería *abb*, de longitud 3, comenzando en la posición 1, como:

*abbcababccabbac.*

Observe que, la extensión reproducible siempre tiene longitud menor que la longitud de  $S$  menos  $j$ :  $\ell(S) - j$ , ya que ninguna cadena que empiece en la posición  $j + 1$  puede ser más larga que eso. Además, observe que la extensión reproducible puede describirse como el par: "posición dónde comienza" y "longitud".

No obstante, note que el cómputo de la extensión reproducible es perfectamente programable: sabemos que a lo sumo es necesario procesar  $\ell(S) - j$  símbolos de la cadena  $S$ , para cada una de las  $j$  primeras posiciones de la cadena, por definición: un total de  $(\ell(S) - j) \times j$  símbolos tienen que ser revisados, en el peor de los casos, antes de terminar el cómputo. El algoritmo ?? ilustra el procedimiento más directo para encontrar la extensión producible de una cadena  $S$ , dado un entero  $j$ ,  $j \leq \ell(S)$ .

---

**Algorithm 1:** Extension\_reproducible
 

---

**Input:** String  $S$ , int  $j$ 
**Output:** Descripción de la extensión reproducible

```

1  $extens = [ ]$ ;
2  $i = 1$ ;
3 while  $i \leq j$  do
4    $ext = 0$ ;
5   while  $ext \leq \text{len}(S) - j$  do
6     if  $S[i : i + ext - 1] == S[j + 1 : j + ext]$  then
7        $ext++$ ;
8     else
9       break;
10   $extens.append\_pair(ext, i)$ ;
11   $i++$ ;
12  $ext\_rep = \text{max}(extens, \text{ord} = \text{"Dictionary"})$ ;
13 return  $ext\_rep$ ;

```

---

En resumen, la extensión reproducible puede codificarse como un par de números enteros que referencian la posición y el tamaño de la extensión reproducible. No en vano, lo anterior constituye una codificación de diccionario adaptativo, donde cada cadena puede expresarse como la posición y la longitud de su extensión reproducible, si la tiene.

Ahora bien, habiendo entendido lo qué es y cómo se codifica la extensión reproducible para cualquier cadena  $S$  y posición  $j$ , es posible entender en qué consiste la codificación LZ77: básicamente, emplea una ventana móvil de tamaño de fijo,  $n$ , la cual se divide en dos partes: búfer diccionario y el búfer extensión. El búfer diccionario contiene aquella porción de la cadena, dentro de la ventana, que ya ha sido procesada por el algoritmo, mientras que el búfer extensión contiene la otra porción de la cadena aún sin codificar. Observe la figura 1. El algoritmo LZ77 codifica la extensión reproducible de la subcadena que empieza en la posición uno

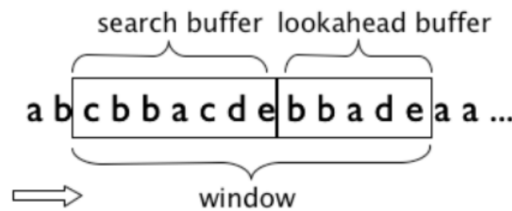


Figure 1: Ventana móvil de tamaño fijo y los búferes de diccionario y extensión. [3]

del búfer extensión, respecto a la subcadena del búfer diccionario. Es decir, aplica la función de extensión reproducible sobre la cadena que está dentro de la ventana en ese momento, para  $j$  igual al tamaño del búfer diccionario. De lo anterior, se obtiene la codificación de la extensión reproducible del búfer extensión, respecto a los patrones del búfer diccionario. Luego, se desplaza la posición de la ventana "extensión reproducible símbolos" a la derecha y repite el mismo proceso mientras que el tamaño del búfer extensión sea mayor que cero. En el proceso, se genera una colección de extensiones reproducibles que codifican la cadena total.

El algoritmo LZ77 genera una palabra código, en cada iteración, en la forma de una terna: "longitud", "posición" y "siguiente símbolo despues de la extensión reproducible". Las dos primeras entradas de la terna codifican la extensión reproducible. Mientras que, "el símbolo despues de la extensión reproducible" almacena el estado de la ventana (del búfer de extensión) en el momento específico en el que se llevó a cabo la codificación. Esto último permitirá al decodificador reconstruir el estado de la ventana en cada instante de tiempo en el que se codificó: al reconstruir la ventana en cada iteración, se recupera la cadena total. Observe la figura 2.

```

while (lookAheadBuffer not empty) {
  get a reference (position, length) to longest match;
  if (length > 0) {
    output (position, length, next symbol);
    shift the window length+1 positions along;
  } else {
    output (0, 0, first symbol in the lookahead buffer);
    shift the window 1 character along;
  }
}

```

Figure 2: Idea básica de la codificación LZ77 [3]

La principal dificultad del algoritmo radica en gestionar apropiadamente el desplazamiento de la ventana. Para esto es necesario tener en cuenta el tamaño,  $n$ , de la ventana, así como el tamaño del búfer extensión,  $L_s$ . Pero además debemos conocer la posición de la ventana en cada iteración: para esto, consideramos la posición de la ventana como el límite derecho y lo denotamos  $win\_pos$ . Note que, en todo momento la ventana se encuentra a la izquierda de  $win\_pos$ , lo cual es conveniente ya que éste alcanzará el último carácter de la cadena antes de que lo haga la ventana, lo cual permitirá definir el criterio de parada para LZ77.

Los detalles técnicos de cómo se desplaza la ventana puede consultarlos en el algoritmo 2.

---

**Algorithm 2:** LZ77

---

**Input:** String  $S$ , int  $n$ , int  $L_s$   
**Output:** Codificación de la cadena

```

1  $winpos = L_s$ ;
2  $B_i = padding(n - L_s)$ ;
3  $B_i += S[1 : L_s]$ ;
4 while  $winpos - L_s < \ell(S)$  do
5   if  $winpos \leq \ell(S)$  then
6      $\ell, p = Extension\_reproducible(B_i, n - L_s)$ ;
7      $\ell = max(1, \ell)$ ;
8      $B_i = B_i[\ell + 1 : n]$ ;
9      $B_i += S[winpos + 1 : winpos + \ell]$ ;
10  else
11     $\ell, p = Extension\_reproducible(B_i[1, \ell(S) - 1], n - L_s)$ ;
12     $\ell = max(1, \ell)$ ;
13     $B_i = B_i[\ell + 1 : \ell(S)]$ ;
14   $winpos += \ell$ ;
15   $output(\ell, p, B_i[1])$ ;

```

---



### 3.2.7 Ejemplo de codificación LZ77

El ejemplo que presentamos a continuación es el ejemplo original que Abraham Lempel y Jacob Ziv utilizaron en su artículo [11] para presentar por primera vez el funcionamiento de su algoritmo de codificación.

Dados los siguiente valores de entrada:

$$S = 001010210210212021021200...$$

$$n = 9$$

$$L_s = 18$$

Llamamos la rutina  $LZ77(S, n, L_s)$  para los valores anteriores: en las líneas uno y dos del algoritmo 2 se definen  $winpos = L_s$  y  $B_i = padding(n - L_s)$ ;

$$B_1 = \boxed{000000000} \downarrow$$

$$S[1 : L_s] = \boxed{001010210} 210212021021200...$$

En la línea tres del algoritmo 2 se lleva a cabo el comando  $B_i+ = S[1 : L_s]$ .

$$B_1 = \boxed{000000000001010210} 210212021021200...$$

En la línea seis del algoritmo 2 se lleva a cabo el comando  $Extension\_reproducible(B_1, n - L_s)$ : la extensión reproducible es  $00$ , de longitud  $\ell = 2$ , empezando en la posición  $p = 9$ , por lo cual, en la línea quince,  $winpos+ = 2$ . A continuación, se retorna la terna  $(2, 9, 1)$ .

Aquí comienza una nueva iteración de  $LZ77(S, n, L_s)$ :

$$B_2 = 000\boxed{000000001010210210} 212021021200...$$

Línea 6:  $Extension\_reproducible(B_2, n - L_s) = (3, 8)$ ,

Línea 14:  $winpos+ = 3$ ,

Línea 15: Se retorna la terna  $(3, 8, 2)$ .

Aquí comienza una nueva iteración de  $LZ77(S, n, L_s)$ :

$$B_3 = 00000000\boxed{000010102102102120} 21021200...$$

Línea 6:  $Extension\_reproducible(B_3, n - L_s) = (7, 7)$ ,

Línea 14:  $winpos+ = \max(1, \ell) = 1$ ,

Línea 15: Se retorna la terna  $(7, 7, 2)$ .

Aquí comienza una nueva iteración de  $LZ77(S, n, L_s)$ :

$$B_4 = 0000000000001010\boxed{210210212021021200} \dots$$

Línea 6:  $Extension\_reproducible(B_4, n - L_s) = (8, 3)$ ,

Línea 14:  $winpos+ = \max(1, \ell) = 1$ ,

Línea 15: Se retorna la terna  $(8, 3, 0)$ .

Cuando queremos codificar cada terna en algún alfabeto de  $\alpha$  símbolos, el número total de  $\alpha$ -bits necesarios para codificar cada terna es

$$L_c = \lceil \log_\alpha(n - L_s) \rceil + \lceil \log_\alpha(L_s) \rceil + 1,$$

$\lceil \log_\alpha(n - L_s) \rceil$  para codificar la posición,  $\lceil \log_\alpha(L_s) \rceil$  para codificar la longitud y 1 para codificar el "símbolo siguiente".

### 3.2.8 Versión funcional

A continuación, podrá encontrar una versión realmente funcional del algoritmo LZ77, que acabamos de ver. Lo importante, a continuación, es que usted sea capaz de notar por usted mismo los tipos de dato que son cada una de las variables.

```

1 def rep_extension(self, search: bytearray, lookahead: bytearray) -> Sequence[int]:
2     n: int = self._n
3     Ls: int = self._Ls
4     pos: int = -1
5     size: int = 0
6     char: chr = ''
7     for prefixsize in range(1, min(n-Ls, len(lookahead))):
8         prefix: str = lookahead[:prefixsize]
9         p: int = search.rfind(prefix, 0, (n-Ls)+prefixsize-1)
10        if p >= 0:
11            pos = p
12            size = prefixsize
13            char = lookahead[size]
14        else:
15            break
16    return pos, size, char

```

```

1 def codify_cad(self, cad: bytearray) -> None:
2     n: int = self._n
3     Ls: int = self._Ls
4     symb: int = self._symb
5     pcad: bytearray = inipad(symb, n-Ls, cad)
6     i: int = 0
7     while i < len(pcad)-(n-Ls):
8         prefix: Sequence[int] = self.rep_extension(pcad[i:i+n], pcad[i+n-Ls:i+n])
9         pos: int = prefix[0]
10        size: int = prefix[1]
11        if (pos >= 0 and size > 1):
12            self._compressed_string.append(True)
13
14            bin_code: str = self.codify_word(pos, size)
15
16            for bit in bin_code:
17                if bit == '1':
18                    self._compressed_string.append(True)
19                else:
20                    self._compressed_string.append(False)
21            i += size
22        else:
23            self._compressed_string.append(False)
24            self._compressed_string.frombytes(bytes([pcad[i+n-Ls]]))
25            i += 1
26    return None

```

**Algorithm 3:** Extension\_reproducible

---

**Input:** String  $S$ , int  $j$   
**Output:** Descripción del prefijo más largo que comienza en la posición  $j$  de  $S$

```

1  $prefixesDescrps = [ ]$ ;
2  $i = 1$ ;
3 while  $i \leq j$  do
4    $prefixLength = 0$ ;
5   while  $prefixLength \leq len(S) - j$  do
6     if  $S[i, i + prefixLength - 1] == S[j + 1, j + prefixLength]$  then
7        $prefixLength++$ ;
8     else
9       break;
10   $prefixesDescrps.append(< prefixLength, i >)$ ;
11   $i++$ ;
12 return  $\max(prefixesDescrps, ord = \text{"Dictionary"})$ ;
```

---

### 3.3 Calculando la Entropía Relativa por medio de la compresión de archivos

Ahora, procedemos a derivar un criterio de desempeño, la proporción de compresión, obtenible por el algoritmo LZ77. Así mismo, proponemos una cota para este desempeño. Esto lo hacemos para el caso en el que se considera un mensaje de la fuente de longitud  $(n - L_s)$ , donde  $n$  es la longitud del buffer y  $L_s$  es la longitud máxima de la palabra. Al final, notaremos que la cota obtenida para este caso particular aplica para mensajes de longitud exactamente igual que  $n - L_s$  y mayor, gracias a la forma (y generalidad) de la cota que se obtiene. En todo caso, la cota obtenida será significativa: para las palabras de longitud menor que  $n - L_s$  no tiene sentido definirla, pues en ese caso se emplearían menos bits para codificar la palabra enviando la palabra en sí misma que codificada con el formato LZ77.

#### 3.3.1 Definiendo el criterio de desempeño

Por construcción, sabemos que toda cadena  $S \in \sigma\{n - L_s\}$  se descompone en una colección de subcadenas,  $\{S_i\}_{i=0}^{N(S)}$ , de longitudes máximas  $L_s$  (variables) durante el procedimiento de codificación LZ77. Luego, se mapean cada una de estas a su respectiva palabra código de longitud fija  $L_c$ . Asociado a cada una de estas asignaciones está una cantidad continua y positiva, que da cuenta de qué tanto la codificación está comprimiendo:

$$\frac{L_c}{\ell(S_i)}.$$

Esta cantidad es estrictamente mayor que 1 si y solo si la longitud de la palabra código es estrictamente mayor que la palabra codificada. Así, cuando  $L_c/\ell(S_i) < 1$  diremos que la codificación está comprimiendo la subcadena  $S_i$ . En este orden de ideas, también podemos saber, de manera global, si la codificación está comprimiendo la palabra  $S$ : verificando si

$$\rho(S) = \frac{L_c \cdot N(S)}{\left(\sum_{i=1}^{N(S)} \ell(S_i)\right)} = \frac{L_c}{n - L_s} \cdot N(S) < 1.$$

No en vano, este concepto posibilita definir un criterio para evaluar el rendimiento de la codificación LZ77 sobre una fuente en particular: definiendo

$$\rho(\sigma) = \frac{L_c}{n - L_s} \cdot N \quad \text{con} \quad N = \max_{S \in \sigma\{n - L_s\}} N(S),$$

como la "tasa de compresión máxima alcanzable" por el algoritmo para la fuente  $\sigma$ . En todo caso, esta tasa es óptima, pues es la tasa mínima alcanzable para cualquier longitud de palabra que se reduce con el formato LZ77 (mínimo denominador,  $n - L_s$ ), al tiempo que es la máxima entre las palabras de longitud  $n - L_s$ . Como tal, merece la pena estudiar este valor y saber hasta qué punto puede mejorarse, o en otras palabras, acotarse superiormente, ya que así también acotaríamos superiormente la "tasa de compresión máxima alcanzable" para todas las palabras de longitud mayor o igual que  $n - L_s$ : mientras mayor sea la longitud de la palabra, menor será el denominador, y por consiguiente, el ratio de compresión.

### 3.3.2 Acotando la tasa de compresión máxima alcanzable

Para acotar la tasa de compresión máxima alcanzable, las cantidades  $L_c$  y  $(n - L_s)$  son fijas, únicamente  $N$  puede variar. Por lo tanto, utilizaremos a  $N$  para inducir esa cota superior: expresando a  $N$  de una manera que pueda acotarse, y así aumentar el valor de  $N$  a  $N' > N$ .

Considere  $Q \in \sigma\{n - L_s\}$  tal que  $N(Q)$ , el número de subcadenas en las cuales  $Q$  es descompuesto por el algoritmo LZ77, es igual a

$$N = \max_{S \in \sigma\{n - L_s\}} N(S).$$

De modo que, el algoritmo descompuso a  $Q$  en el máximo número de subcadenas en las que descompuso cualquier palabra de longitud  $n - L_s$ , como  $Q = Q_1 Q_2 \cdots Q_N$ .

Ahora bien, para todo  $m$ , con  $1 \leq m \leq L_s$ , definimos  $K_m$ : el número de subcadenas,  $Q_i$ , de tamaño  $m$  e  $1 \leq i \leq N - 1$ . Con esta definición, podemos expresar las cantidades  $N$  y  $(n - L_s)$  en términos de los  $K_m$  como:

$$N = 1 + \sum_{m=1}^{L_s} K_m$$

y

$$n - L_s = \ell(Q_N) + \sum_{m=1}^{L_s} m \cdot K_m.$$

Ahora bien, esta representación de  $N$  y  $n - L_s$  en términos de los  $K_m$  no es casual: los  $K_m$  pueden acotarse superiormente, de hecho, cumplen la siguiente propiedad, que es consecuencia directa de la forma en la que se contruyen las subcadenas,  $Q_i$ , durante el proceso de codificación LZ77.

**Lemma 3.1.** *Dado  $Q = Q_1 \cdots Q_N$ ,*

$$\text{si } \ell(Q_i) = \ell(Q_j) < L_s, \text{ para cualesquiera } i < j < N, \text{ entonces } Q_i \neq Q_j.$$

*Proof.* Supongamos que  $\ell(Q_i) = \ell(Q_j) < L_s$  para  $i < j < N$ .

En la  $j$ -ésima iteración del algoritmo de codificación LZ77, durante el paso número dos, la subcadena  $Q_j$  está siendo determinada: se origina en la posición  $n - L_s + 1$ , de la ventana, y tiene longitud  $\ell(Q_j) < L_s$ . Mientras tanto, la subcadena  $Q_i$  se origina en la posición  $n - L_s$ , o antes, y tiene longitud  $\ell(Q_i) < L_s$ .

Ahora bien, dado que  $\ell(Q_j) < L_s$ , entonces la longitud del prefijo más largo de la subcadena  $Q_j$ , ella misma, tendrá longitud  $\ell(Q_j) < L_s$ . De modo que la longitud del prefijo más largo de cualquier subcadena  $Q_j$  es siempre menor o igual que su longitud,  $\ell(Q_j)$ , y distinto a la longitud fija máxima,  $L_s$ . Por consiguiente, la subcadena más larga que se origina antes de la posición en la que se origina  $Q_j$ ,  $n - L_s + 1$ , y que a su vez es un prefijo de  $Q_j$  tiene longitud  $\ell(Q_j) - 1$ . Es decir, la subcadena  $Q_j$  no coincide con cualquier otra subcadena de tamaño  $\ell(Q_j)$ , particularmente  $Q_i$ , en al menos su último carácter. Por lo tanto,  $Q_j \neq Q_i$ .  $\square$

**Corollary 3.1.1.**

$$K_m \leq \sigma(m) \text{ para todo } 1 \leq m \leq L_s - 1.$$

*Proof.* Debido a que sabemos, por el lema anterior, que ninguna cadena de longitud menor que  $L_s$  aparece más de una vez, entonces el número de subcadenas de longitud  $m \leq L_s - 1$  en la descomposición de  $Q$ ,  $K_m$ , puede ser a lo sumo igual al número de subcadenas de longitud  $m$  que puede generar la fuente,  $\sigma(m)$ .  $\square$

Así, podemos sobrestimar el valor de  $N$  por medio de los  $K_m$ :  $K_m \leq \sigma(m) = \alpha^{\log_\alpha \sigma(m)} = \alpha^{mh(m)} \leq \alpha^m$ , el número total de cadenas distintas de longitud  $m$  que pueden construirse con un alfabeto de  $\alpha$  símbolos. De aquí también se sigue que  $\sigma(m) \leq \sigma(m+1)$  para todo  $1 \leq m \leq L_s - 1$ . De esta manera, se puede verificar que

$$N \leq \left( \sum_{m=1}^{L_s-1} K'_m \right) + K'_{L_s}, \quad \text{con} \quad K'_{L_s} = \left\lceil \frac{1}{L_s} \left( n - L_s - \sum_{m=1}^{L_s-1} m K'_m \right) \right\rceil$$

donde

$$K'_m = \begin{cases} \alpha^m & \text{si } 1 \leq m \leq \lfloor \log_\alpha (\sigma(L_s - 1)) \rfloor \\ \sigma(L_s - 1) & \text{si } \lfloor \log_\alpha (\sigma(L_s - 1)) \rfloor \leq m \leq L_s - 1 \end{cases}$$

No obstante,  $m \leq \lfloor \log_\alpha (\sigma(L_s - 1)) \rfloor$  implica que  $\alpha^m \leq \alpha^{\lfloor \log_\alpha (\sigma(L_s - 1)) \rfloor} \leq \alpha^{\log_\alpha (\sigma(L_s - 1))} = \sigma(L_s - 1)$ . Así que  $K_m$  se sobrestima por medio de  $\alpha^m$  siempre y cuando  $\alpha^m \leq \sigma(L_s - 1)$ . Mientras que, cuando  $\alpha^m > \sigma(L_s - 1)$ ,  $K_m$  se sobrestima por medio de  $\sigma(L_s - 1)$ : es importante garantizar que  $K'_m$  no sobrepasa el valor de  $\sigma(m)$ , de lo contrario, se estaría particionando a  $Q$  en cadenas que  $\sigma$  no puede generar. Así, garantizamos que  $K_m \leq K'_m$  para todo  $1 \leq m \leq L_s - 1$ , al tiempo que la correctitud del algoritmo.

Mientras tanto, la porción restante de  $Q$ , de longitud  $(n - L_s - \sum_{m=1}^{L_s-1} m K'_m)$ , se divide en subcadenas de longitud  $L_s$ , por lo que se tienen  $K'_{L_s}$  cadenas de longitud  $L_s$ . No obstante,  $K'_{L_s}$ , el número de cadenas de longitud  $L_s$  pudiera ser negativo o cero, y por tanto menor que  $K_m$  (el número original de cadenas de longitud  $L_s$ ). En realidad, lo que se hizo fue reemplazar las subcadenas más largas, que ocupan un mayor espacio con un menor número de ellas, por subcadenas más cortas que maximizan el número de subcadenas en las que se particiona  $Q$ .

De hecho, acabamos de construir el supremo del número de subcadenas en las que se puede particionar cualquier cadena de longitud  $n - L_s$  generada por la fuente, por medio del algoritmo de codificación LZ77: vimos que era mayor o igual al máximo número de subcadenas en las que el algoritmo particionó a  $Q$ .

**Lemma 3.2.**

$$n = \sum_{m=1}^{\lambda} m \alpha^m + \sum_{m=\lambda+1}^{L_s-1} m \sigma(L_s - 1) + L_s (N_{L_s} + 1)$$

donde

$$N_{L_s} = \sum_{m=1}^{\lambda} (L_s - 1 - m) \alpha^m + \sum_{m=\lambda+1}^{L_s-1} (L_s - 1 - m) \sigma(L_s - 1)$$

y

$$\lambda = \lfloor \log_\alpha (L_s - 1) \rfloor$$

*Proof.* Primero, observe que

$$\begin{aligned}
\sum_{m=1}^{L_s} mK'_m &= \left( \sum_{m=1}^{L_s-1} mK'_m \right) + L_s \cdot K'_{L_s} \\
&= \left( \sum_{m=1}^{L_s-1} mK'_m \right) + L_s \cdot \left[ \frac{1}{L_s} \left( n - L_s - \sum_{m=1}^{L_s-1} mK'_m \right) \right] \\
&= \left( \sum_{m=1}^{L_s-1} mK'_m \right) + \left[ L_s \cdot \frac{1}{L_s} \left( n - L_s - \sum_{m=1}^{L_s-1} mK'_m \right) \right] \\
&= \left( \sum_{m=1}^{L_s-1} mK'_m \right) + \left[ \left( n - L_s - \sum_{m=1}^{L_s-1} mK'_m \right) \right] \\
&= \left( \sum_{m=1}^{L_s-1} mK'_m \right) + \left( n - L_s - \sum_{m=1}^{L_s-1} mK'_m \right) \\
&= n - L_s.
\end{aligned}$$

Efectivamente, podemos escribir la longitud de la cadena en términos de los nuevos  $K'_m$ , para  $1 \leq m \leq L_s$ .

No en vano, por como definimos a  $K'_m$ , se sigue que

$$\begin{aligned}
n - L_s &= \sum_{m=1}^{L_s} mK'_m \\
&= \left( \sum_{m=1}^{L_s-1} mK'_m \right) + L_s \cdot K'_{L_s} \\
&= \left( \sum_{m=1}^{\lambda} m\alpha^m + \sum_{m=\lambda+1}^{L_s-1} m\sigma(L_s - 1) \right) + L_s \cdot K'_{L_s}.
\end{aligned}$$

donde  $\lambda = \lfloor \log_{\alpha}(L_s - 1) \rfloor$ . De donde se sigue que

$$\begin{aligned}
n &= \left( \sum_{m=1}^{\lambda} m\alpha^m + \sum_{m=\lambda+1}^{L_s-1} m\sigma(L_s - 1) \right) + L_s \cdot K'_{L_s} + L_s \\
&= \left( \sum_{m=1}^{\lambda} m\alpha^m + \sum_{m=\lambda+1}^{L_s-1} m\sigma(L_s - 1) \right) + L_s \cdot (K'_{L_s} + 1)
\end{aligned}$$

□

Con esto, ya estamos listos para formular el resultado principal de esta sección, de cual se desprende la cota superior de la tasa de compresión máxima alcanzable:

**Theorem 3.3.**

$$(n - L_s) - (L_s - 1)N' = 0$$

*Proof.*

□

## Corollary 3.3.1.

$$\rho(\sigma) \leq \frac{L_c}{L_s - 1}$$

## References

- [1] Dario Benedetto, Emanuele Caglioti, and Vittorio Loreto. “Language Trees and Zipping”. In: Physical Review Letters 88.4 (2002). DOI: 10.1103/PhysRevLett.88.048702.
- [2] G. J. Chaitin. “A theory of program size formally identical to information theory”. In: Journal of the ACM 22.3 (1975), pp. 329–340. DOI: 10.1145/321892.321894.
- [3] K.Lindgren. Information theory for complex systems. Complex systems group. Department of Energy and Environment, 2014. ISBN: 9780198520115.
- [4] A. N. Kolmogorov. “Three approaches to the quantitative definition of information”. In: Problems in Information Transm 1.1 (1965), pp. 1–7.
- [5] Ming Li and Paul Vitányi. An Introduction to Kolmogorov Complexity and Its Applications. Texts in computer science. Springer, 2019. ISBN: 978-3-030-11297-4.
- [6] Toshiko Matsumoto. “Biological Sequence Compression Algorithms”. In: Genome Informatics 11 (2000), pp. 43–52. DOI: 10.11234/GI1990.11.43.
- [7] C. E. Shannon. “A mathematical theory of communication”. In: The Bell System Technical Journal 27.3 (1948), pp. 379–423. DOI: 10.1002/j.1538-7305.1948.tb01338.x.
- [8] R. J. Solomonoff. “A formal theory of inductive inference”. In: Information and Control 7 Parts 1 and 2.1-22 (1964), pp. 224–254. DOI: [https://doi.org/10.1016/S0019-9958\(64\)90223-2](https://doi.org/10.1016/S0019-9958(64)90223-2).
- [9] Andreia Teixeira et al. “Entropy Measures vs. Kolmogorov Complexity”. In: Entropy 13.3 (2011), pp. 595–611. ISSN: 1099-4300. DOI: 10.3390/e13030595. URL: <https://www.mdpi.com/1099-4300/13/3/595>.
- [10] Paul M.B. Vitányi. “How Incomputable Is Kolmogorov Complexity?” In: Entropy 22.4 (2020). ISSN: 1099-4300. DOI: 10.3390/e22040408. URL: <https://www.mdpi.com/1099-4300/22/4/408>.
- [11] Jacob Ziv and Abraham Lempel. “A Universal Algorithm for Sequential Data Compression”. In: IEEE TRANSACTIONS 23.3 (1977), pp. 337–343. DOI: 10.1109/TIT.1977.1055714.

Participantes:

Firma:

\_\_\_\_\_  
Álvarez Cabrera, Carlos Eduardo. Ph.D.  
Tutor, Matemáticas Aplicadas y Ciencias de la Computación  
Tuesday 18<sup>th</sup> May, 2021

Firma:

\_\_\_\_\_  
Hernández Ramírez, Esteban.  
Estudiante, Matemáticas Aplicadas y Ciencias de la Computación  
Tuesday 18<sup>th</sup> May, 2021