Algoritmo LZ77: Comprensión e implementación en JAVA.

Kevin Andrés Borda Penagos - 20171020088
Mateo Yate Gonzalez - 20171020087

Universidad Distrital Francisco José de Caldas.
Facultad de Ingeniería.
Proyecto Curricular de Ingeniería de Sistemas.
Redes de Comunicaciones II.
Profesor: José Martín Delgado García.
14/08/20.

UNIVERSIDAD DISTRITAL
FRANCISCO JOSÉ DE CALDAS

‘Algoritmo sin pérdida de datos, el cual trabaja enfocado en encontrar coincidencias en la cadena’

Características.

- Es un algoritmo sin pérdida de datos.
- Es dinámico (ventanas deslizables).
- Sencillo y eficiente.
- Combinado con la codificación de Huffman, producen el algoritmo DEFLATE usado por GZIP y WinZIP
- Aplicado en imágenes en formato GIF, Compress Unix y DESINFLE (PNG y Postal)

Historia

- Abraham Lempel y Jacob Ziv (1977)
- Modificaciones
  - 1978 LZ78
  - 1982 LZSS
  - 1984 LZW
  - 1996 LZMA

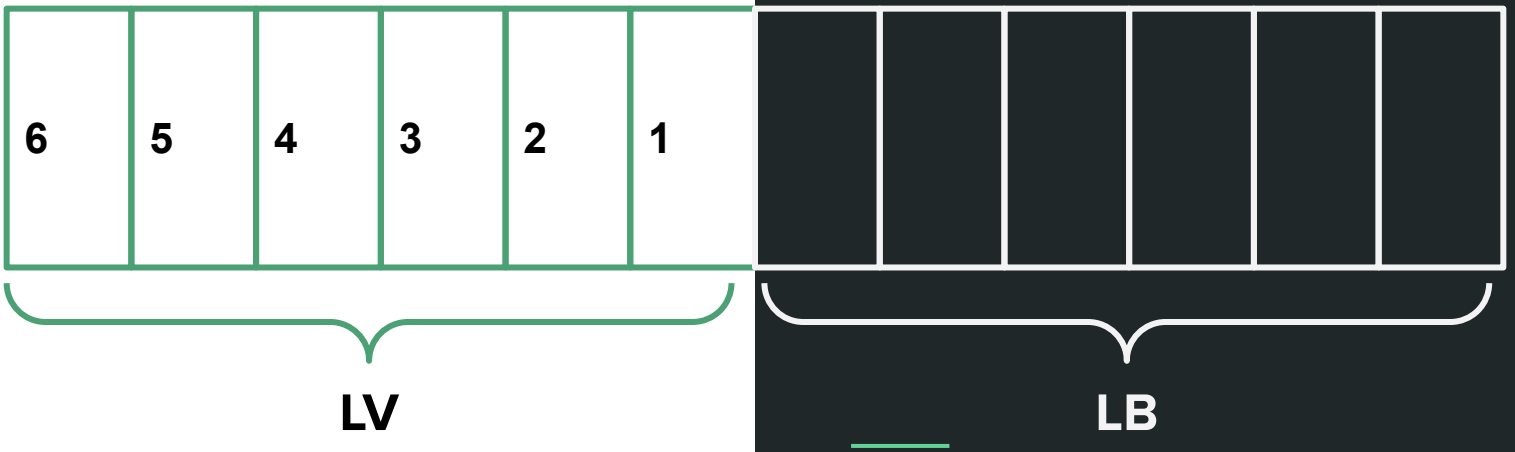
Codificación

Ventana

- Listado donde se buscarán las coincidencias, y será la referencia para la decodificación.

Buffer

- Puente entre la ventana y la cadena, será el encargado de buscar coincidencias.

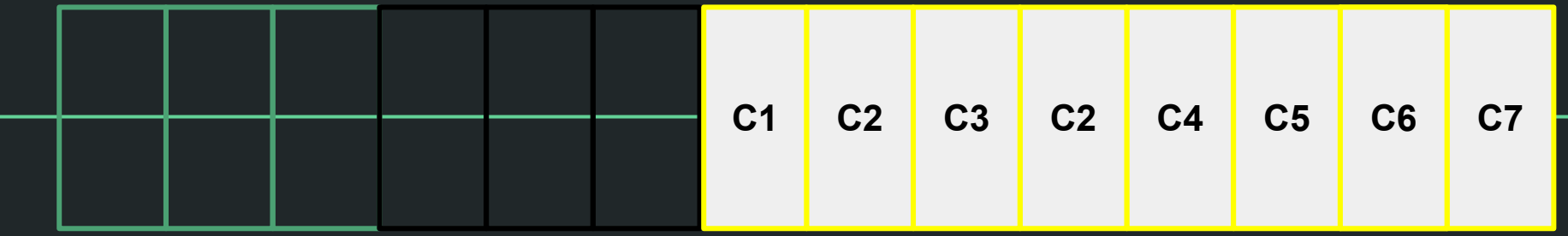


Antes de empezar

- Posición coincidencia: Posición en la ventana que coincide con la cabecera del Buffer.
- Tamaño: Si los siguientes caracteres siguen siendo iguales se toma el largo, hasta que aparezca uno que no coincida con la ventana.
- Carácter de ruptura: Carácter del buffer que no está en la ventana.
- Ternas: <PC, T, CR>.
- Final marcado por EOF.

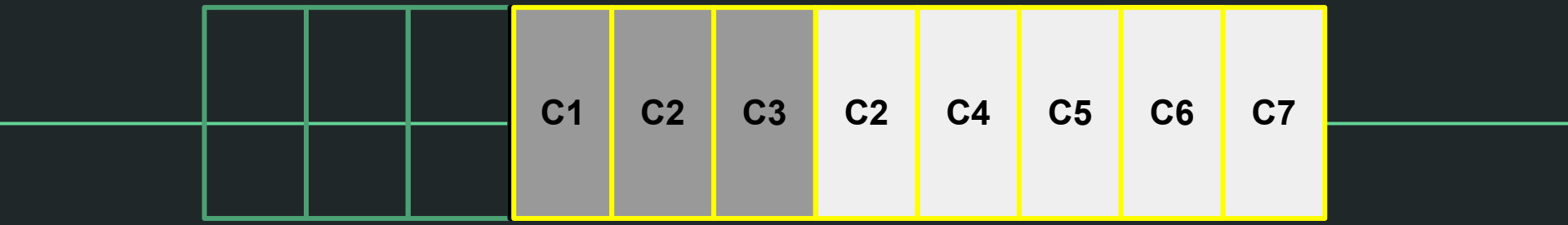
¿Como trabaja?

1. Definir tamaño buffer y ventana



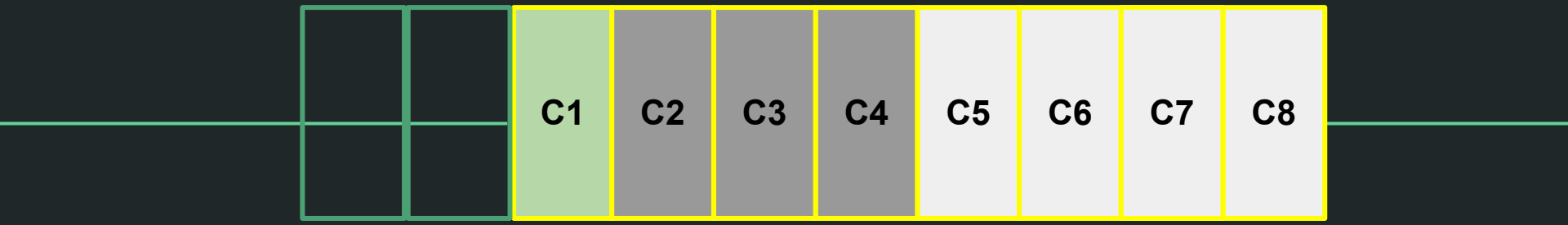
¿Como trabaja?

2. Inicializar Buffer

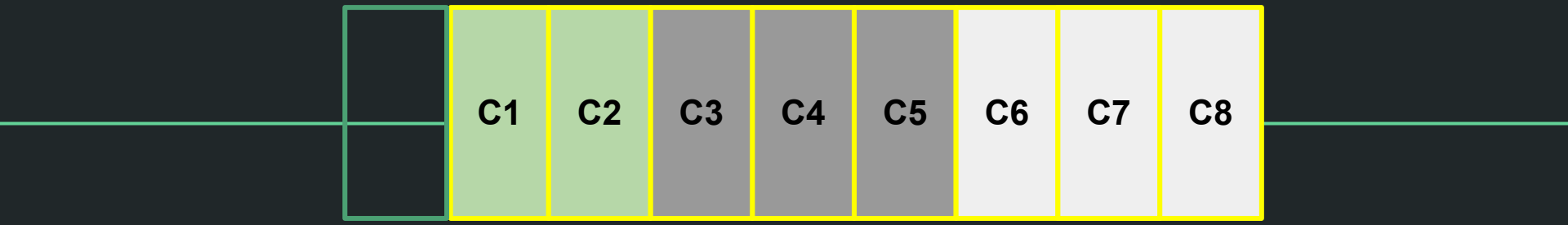


¿Como trabaja?

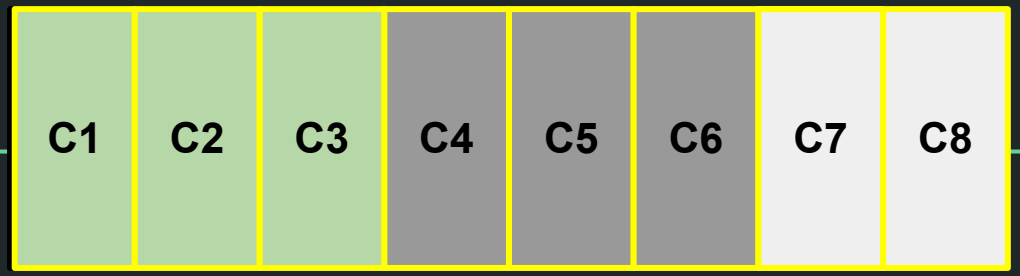
3. Empezar a desplazar



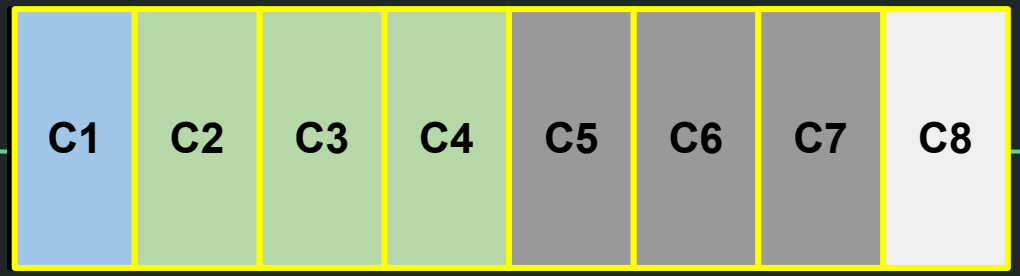
¿Como trabaja?



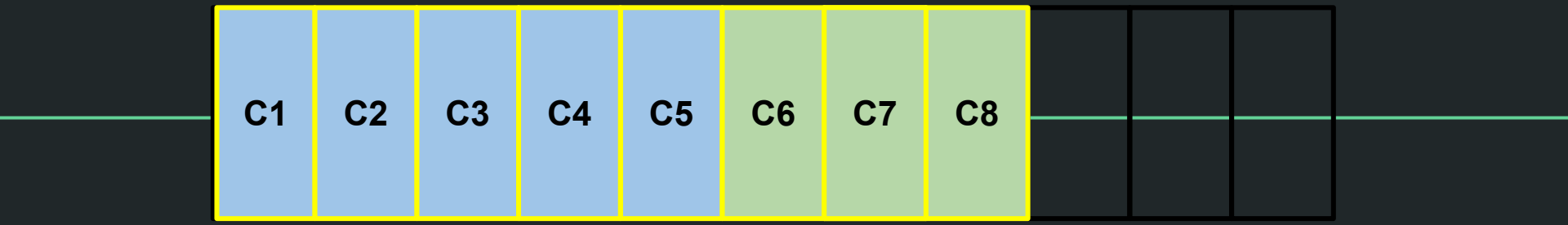
¿Como trabaja?



¿Como trabaja?

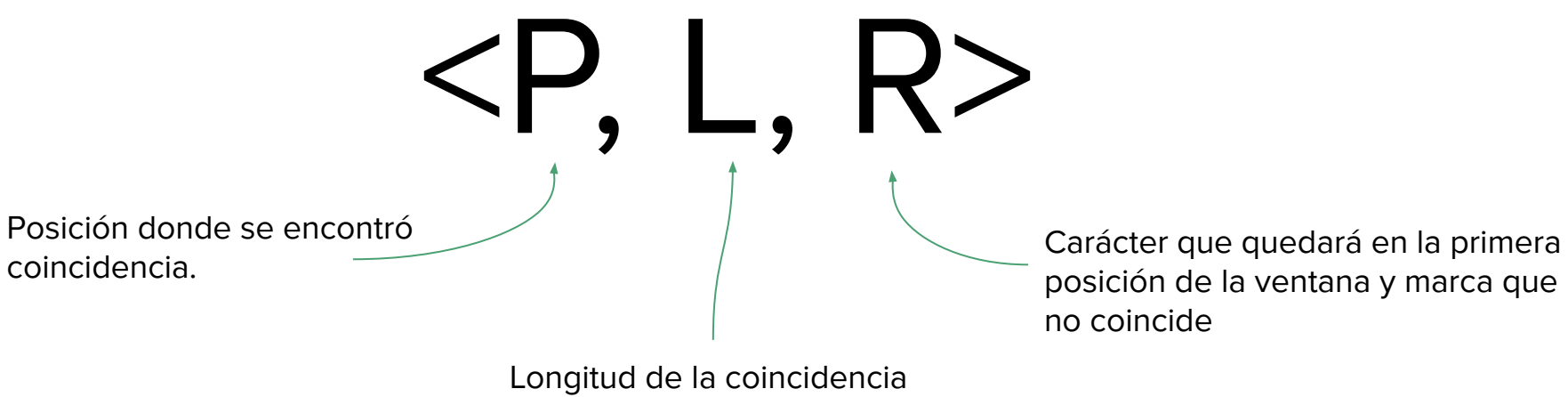


¿Como trabaja?



Representación de resultados.

Para la representación lo haremos mediante ternas.



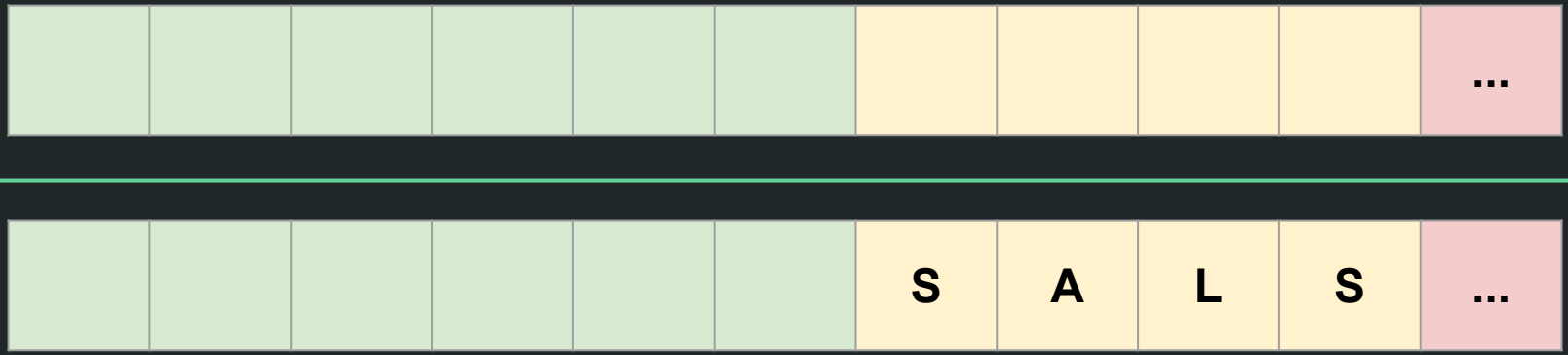
Ejemplo

SALSA SALADA

Ventana = 6

Buffer = 4

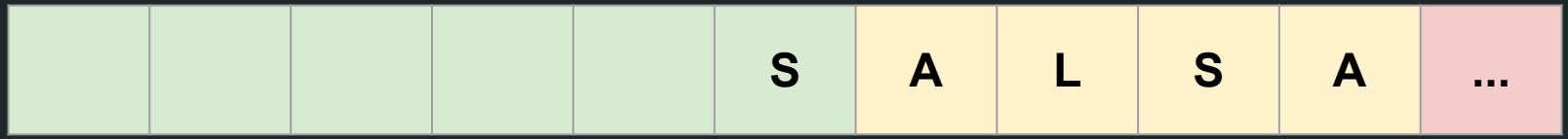
SALSA SALADA’

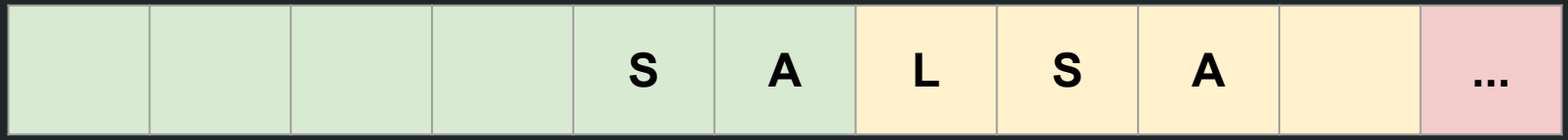


<0,0,S>

‘SALSA SALADA’

<0,0,S>



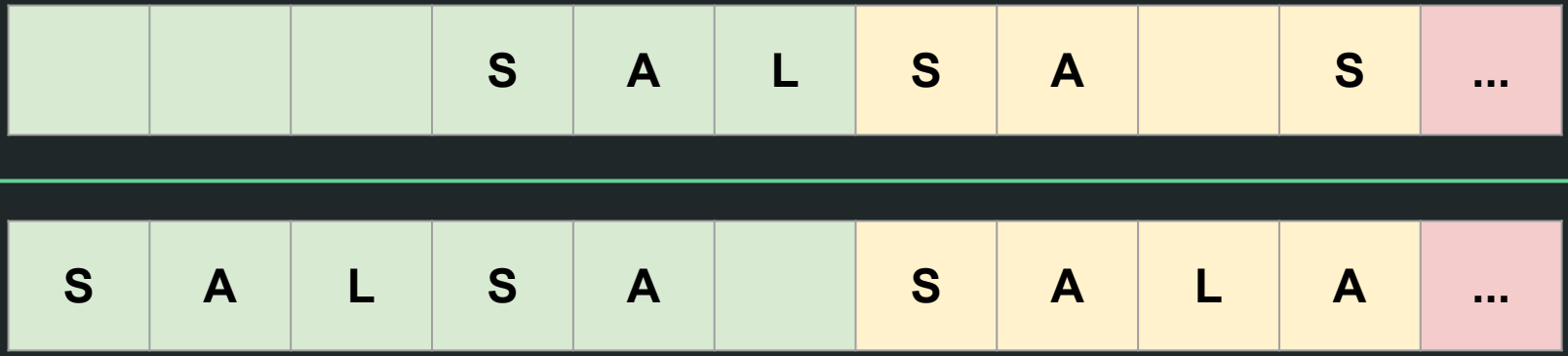


<0,0,A>

<0,0,L>

‘SALSA SALADA’

<0,0,S>,<0,0,A>,<0,0,L>

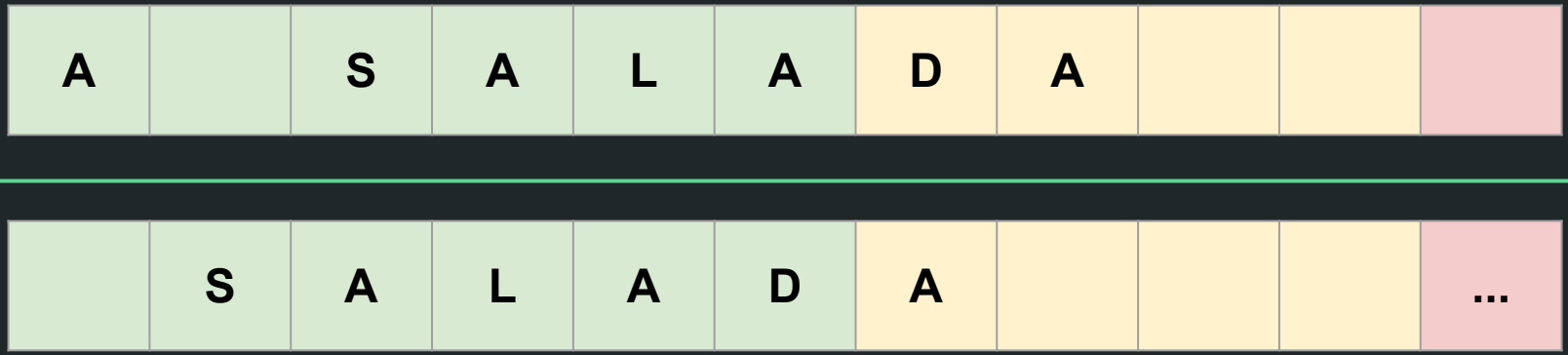


<3,2,‘ ’>

<6,3,A>

‘SALSA SALADA’

<0,0,S>,<0,0,A>,<0,0,L>,<3,2,‘ ’>,<6,3,A>



<0,0,D>

<4,1,EOF>

‘SALSA SALADA’

<0,0,S>,<0,0,A>,<0,0,L>,<3,2,‘ ’>,<6,3,A>,<0,0,D>,<4,1,EOF>

<0,0,D>

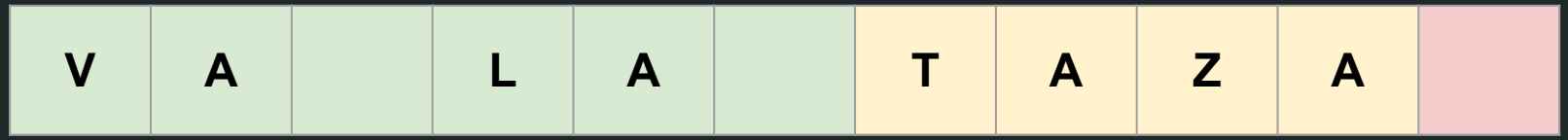
<4,1,EOF>

Caso especial en la codificación.

Todo el Buffer coincide con la ventana.

‘CARLA LAVA LA TAZA’

Tamaño de las las ventanas:
Ventana = 6
Buffer = 4



<5,4, ¿?>

<5,4,‘ ’>

Decodificación

¿Qué debe saberse?

- Se realiza con base en una serie de ternas dadas.
- No importa el tamaño del buffer o de la ventana en el que el texto fue codificado.
- El número de iteraciones es menor a la longitud de la cadena decodificada.
- Suele ser mucho más sencilla que el proceso de codificación.

¿Cómo trabaja?

Teniendo en cuenta el formato de terna expuesto anteriormente (<P, L, R>), se empieza desde la terna inicial y para cada una de las ternas se toman las siguientes condiciones:

- Si el primer valor de la terna (P) es cero: Se añade el valor de R al texto decodificado.
- Si el valor de P es distinto de cero: Se toma la posición P de la coincidencia * en la palabra decodificada, y teniendo en cuenta la longitud de la misma (L), la cadena de coincidencia se copia al final del texto decodificado y después se le añade el valor de R ai final.

*** Importante: Esta posición es relativa, es decir, si se codificó leyendo las coincidencias de izquierda a derecha, así mismo se debe decodificar.**

Ejemplo

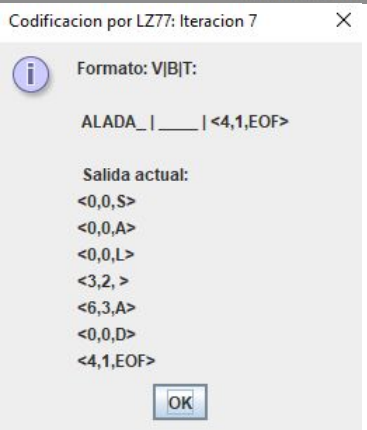
SALSA SALADA

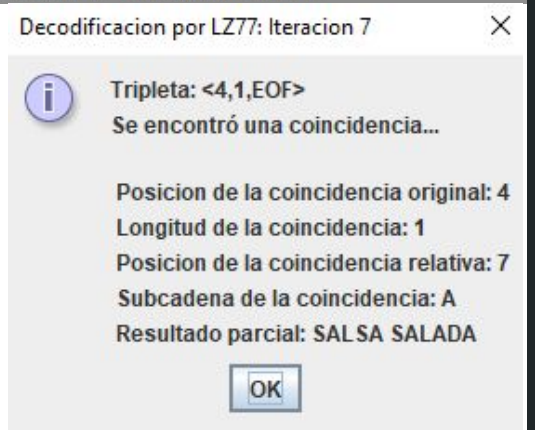
<0,0,S>, <0,0,A>, <0,0,L>, <3,2,’ ’>, <6,3,A>, <0,0,D>, <4,1,EOF>

‘SALSA SALADA’

Terna codificada	Coincidencia + siguiente carácter	Cadena decodificada
<0,0,S>	S	S
<0,0,A>	A	SA
<0,0,L>	L	SAL
<3,2,’ ’>	SA_	SALSA
<6,3,A>	SALA	SALSA SALA
<0,0,D>	D	SALSA SALAD
<4,1,EOF>	A	SALSA SALADA

Implementación vía JAVA.





- Se desarrolló un aplicativo en el lenguaje de programación Java para ilustrar el funcionamiento del algoritmo.
- Este programa es capaz de codificar desde un campo de texto y decodificar desde un archivo con tuplas separadas por renglones.
- El proyecto se encuentra alojado en el repositorio: https://github.com/mateoyateg/CodificadorLZ77.

Documento investigativo

Algoritmo LZ77: Comprensión e implementación en un aplicativo empleando el lenguaje de programación JAVA.

Mateo Yate Gonzalez, Kevin Andrés Borda Penagos.
Facultad de Ingeniería, Universidad Distrital Francisco José de Caldas.
Bogotá D.C., Colombia.
myateg@correo.udistrital.edu.co
kabordap@correo.udistrital.edu.co

Resumen - El presente documento ilustra conceptos claves relacionados con el algoritmo de codificación y decodificación de datos LZ77, esto por vía de una investigación realizada para lograr entender el funcionamiento del algoritmo mismo, para así llevar a cabo la implementación del funcionamiento de codificación y decodificación en un aplicativo en Java que muestre de una manera más didáctica y clara ambos procesos descritos anteriormente.

Índice de Términos (*Keywords*) - LZ77, algoritmos de codificación y decodificación de datos, codificación, Java.

I. INTRODUCCIÓN

El presente documento presenta una investigación realizada acerca del surgimiento del algoritmo de codificación y decodificación de datos LZ77, brindando un marco del contexto en que este surgió, su definición misma y sus convenciones utilizadas por los distintos autores, así como sus características técnicas, algunos ejemplos de funcionamiento y finalmente la implementación realizada en el lenguaje de programación Java.

II. CONTEXTO HISTÓRICO

Antes de empezar a hablar más a profundidad sobre el algoritmo LZ77, es de amplia importancia conocer el contexto histórico en el que este se gestó, además de su historia misma.

Para iniciar, se puede decir que este algoritmo no es contemporáneo a las codificaciones populares como Hamming o Huffman, por el contrario es posible afirmar que LZ77 es más reciente.

Sus inicios datan del año 1977 cuando Abraham Lempel y Jacob Ziv presentaron este modelo como un compresor sin pérdida de datos, o como también es conocido, un compresor de texto.

El nombre del algoritmo tiene sus raíces en lo anteriormente descrito y es que, las iniciales de los 2 apellidos, concatenadas con los 2 últimos dígitos del año que se presentó generan como resultado "LZ77", por ende, este acrónimo tiene su origen de manera más específica en Lempel, Ziv y 1977.

Su gran impacto puede verse caracterizado por el hecho de la implementación de una ventana deslizable la cual no se encontraba en algoritmos primitivos y de la que se hablará en secciones posteriores del presente informe. Es de resaltar que esta implementación mejoró notoriamente la efectividad en la compresión. [1]

La propuesta inicial que hicieron los autores sólo ha sufrido una modificación y esta está dada en su implementación al momento de codificar archivos. En 1982 James Storer y Thomas Szymanski hicieron un pequeño cambio y en vez de tomar los 8 bits de una posición, se colocaba una bandera (o posición de coincidencia, como se conoce hoy en día) que sería el punto a llamar, la cual sólo ocuparía un bit. Esto hacía más óptimo el proceso, por esta razón al día de hoy también se encuentra el algoritmo LZSS, donde como dato curioso, los autores cambian los dígitos del año perteneciente al acrónimo del algoritmo original por sus iniciales. [2]

- Se realizó un documento investigativo donde se trataron aspectos teóricos y se explicó la parte práctica del proyecto.
- Se trataron temas como el contexto histórico, las características técnicas de LZ77, los procesos de codificación y decodificación y demás.
- El documento se encuentra alojado en el mismo repositorio del proyecto.

GRACIAS.

Referencias

- Mayordomo, E. Packing Dimension. Recuperado de: http://webdiis.unizar.es/~elvira/LZ.pdf
- Engineering and Technology History Wiki, 2019. History of Lossless Data Compression Algorithms. [En línea] https://ethw.org/ Disponible en: https://ethw.org/History_of_Lossless_Data_Compression_Algorithms#:~:text=Sliding%20Window%20Algorithms-,LZ77,ratio%20over%20more%20primitive%20algorithms

- Wikipedia. LZSS. [En línea] en.wikipedia.org . Disponible en: https://es.wikipedia.org/wiki/LZSS#:~:text=El%20algoritmo%20de%20compresi%C3%B3n%20lz77,compresores%20que%20utilizan%20algoritmos%20del

- Microsoft., 2020. LZ77 Compression Algorithm. [En línea] https://docs.microsoft.com/ Disponible en: https://docs.microsoft.com/en-us/openspecs/windows_protocols/ms-wusp/fb98aa28-5cd7-407f-8869-a6cef1ff1ccb

- Wiki. LZ77 y LZ78 - LZ77 and LZ78 .[En línea] es.qwe.wiki. Disponible en: https://es.qwe.wiki/wiki/LZ77_and_LZ78#LZ78

Referencias

- Wiki. Lzw - Lempel–Ziv–Welch. [En línea] es.qwe.wiki. Disponible en: https://es.qwe.wiki/wiki/Lempel%E2%80%93Ziv%E2%80%93Welch

- Wiki. lzma - Lempel–Ziv–Markov chain algorithm .[En línea] es.qwe.wiki. Disponible en: https://es.qwe.wiki/wiki/Lempel%E2%80%93Ziv%E2%80%93Markov_chain_algorithm