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]

III. DEFINICIÓN Y CONVENCIONES.

Para empezar a hablar directamente del algoritmo es necesario plantear una definición para el mismo. A pesar de no tener una definición formal, se encontró que muchos autores se refieren a este algoritmo de la siguiente manera: *"El algoritmo de compresión LZ77 se utiliza para analizar los datos de entrada y determinar cómo reducir el tamaño de esos datos de entrada reemplazando la información redundante con metadatos"*. [3]

Se evidencia que esta definición no es tan exacta como debería; aun así, para complementar la misma, es posible afirmar que LZ77 es un algoritmo que se basa en las coincidencias de texto para hacer una compresión de los datos mucho más sencilla y más eficiente a la hora de trabajar con estos, debido a que, en la transmisión de una cadena lo ideal es no repetir datos para así ahorrar tiempo y recursos que podrán ser empleados de una manera óptima más adelante.

Respecto a sus convenciones, se encontraron algunas variantes tal y como se nombró al inicio. Lo que se conoce hoy en día no es la idea de codificación original, en cambio, es una variante presentada en 1982 por los autores Szymanski y Storer, que se basan en usar un punto de coincidencia para trabajar a partir de este; caso contrario a lo que hacía al inicio el algoritmo.

Una variante que no es tan conocida es la que se presentó en 1978, para la cual se encontró la propuesta presentada por los mismos autores un año después, de tal forma que las variaciones entre ambas proposiciones no son muy grandes. Se evidencia una gran diferencia referente a que, el algoritmo LZ78 trabaja con parejas mientras que el algoritmo LZ77 trabaja con ternas, además del reemplazo de la ventana deslizante por un diccionario explícito. Esta propuesto junto con el algoritmo LZ77 fue nombrada como LZ1 y LZ2 por orden cronológico.

De estos algoritmos descritos anteriormente emergen varios algoritmos, como lo son el LZW, LZMA, LZSS y demás. [4]

El algoritmo LZW fue creado en 1984 por los autores originales de LZ77 junto con el estadounidense Terry Welch. Este algoritmo presenta una mejora en cuanto a que evidencia un alto potencial de rendimiento; su uso se ve principalmente en la plataforma Unix y se utiliza en formatos de imágenes GIF. [5]

Finalmente, como adición del algoritmo de Markov la base del LZ77 fue creado el algoritmo LZMA por Igor Pavlov en la época comprendida entre 1996 y 1998. Este algoritmo se empleó por primera vez en el formato de compresión de archivos 7-zip [6].

IV. CARACTERÍSTICAS TÉCNICAS

Como se nombró en la definición inicial, este algoritmo trabaja por coincidencias, pero es objeto de resaltar que además de esta característica, LZ77 cuenta con varios términos propios que son importantes para denotar sus diferencias respecto a algoritmos anteriores.

Se tienen aspectos claves que son un factor diferencial a la hora de definir LZ77, como lo es por ejemplo su dinamismo; este algoritmo tuvo relevancia al momento de su implementación dado que no era un algoritmo estático gracias a su ventana deslizable.

De esta manera, este algoritmo cuenta con una ventana deslizable compuesta por dos secciones que varían dinámicamente y que tienen por nombre "ventana" y "buffer".

Respecto al segundo componente mencionado, el buffer puede verse como un primer elemento de almacenamiento que trabaja como una lista dinámica. Este sirve como un puente entre la ventana y el texto a codificar, su trabajo es tomar por separado los caracteres de la cadena que se está trabajando y a su vez trasladarlos hacia la ventana. Es posible afirmar que esta lista es la más importante porque además es la encargada de notificar si se encuentra una coincidencia o no, pero este tema se tratará a fondo en secciones posteriores del presente informe.

Al referirse a la ventana (que en muchos casos es vista como un complemento del buffer y además es llamada buffer de búsqueda), esta también es vista como una lista del mismo modo que el buffer. Allí es donde se buscarán las coincidencias con respecto al buffer; cabe recalcar que inicialmente, tanto la ventana como el buffer tienen tamaños definidos (el tamaño de la ventana siempre es mayor o igual al del buffer), debido a que no tiene sentido ejecutar el algoritmo siendo ambas magnitudes iguales al largo

de la cadena. Además, es de resaltar que dichas longitudes en la práctica tienen rangos que pueden alcanzar la escala de miles.

La relación de ambas listas (ventana y buffer) genera resultados de manera constante, y es que en cada iteración se genera un resultado distinto. Dichos resultados serán vistos como un conjunto de ternas, donde cada posición de la terna representa algo específico. De esta manera, es posible evidenciar dicha terna como <P,T,C>, donde P se refiere a la posición de la coincidencia, T es el tamaño de la coincidencia y C el carácter que quedará en el inicio de la ventana.

Es de resaltar que el algoritmo LZ77 cuenta con una serie de ventajas y desventajas expuesta a continuación:

Ventajas del algoritmo:

- Es dinámico, por lo cual permite tener una mayor dinámica a la hora de codificar con respecto a los algoritmos estáticos.
- Puede trabajar con cadenas de hasta millones de caracteres y allí se encuentra radicada su fortaleza, ahorrando una importante cantidad de espacio en su codificación.
- Su implementación es muy sencilla y a su vez potente, de aquí es donde se deriva su trabajo en gif y en compresores utilizados habitualmente.

Desventajas del algoritmo:

- Si existe un error puede cambiar totalmente la información, a pesar de ser un algoritmo de no tener pérdida de datos si se hace una mala práctica con el mismo, los resultados pueden ser catastróficos.
- Existen casos en los que el algoritmo no es muy útil como en cadenas que no tengan repetición de caracteres, pero aun así en la práctica esto no es muy común.

V. ¿CÓMO SE UTILIZA EL ALGORITMO LZ77?

El algoritmo LZ77 es un algoritmo capaz tanto de codificar como de decodificar, a continuación se describe su funcionamiento en ambos aspectos.

- Codificación

El comportamiento de este algoritmo es muy sencillo, pero como se recalcó anteriormente, ello no implica que no se deba tener cuidado al momento de emplear el mismo, puesto que si hay un solo espacio de alguna terna que no se escriba bien, el resultado será incorrecto, por lo cual se recomienda tener especial cuidado en este aspecto.

Inicialmente, es importante evidenciar las ya mencionadas “ventanas deslizables” en forma de listas:

Figura 1. Modelo en que serán representadas las ventanas deslizables junto con la cadena.

En este orden es como se encuentra cada una dividida con su respectivo tamaño, y siendo la ventana y el buffer deslizables a lo largo de la cadena, de izquierda a derecha. Cabe resaltar que las posiciones en la ventana van de mayor a menor, es decir su posición 1 está inmediatamente al lado del Buffer.

Su funcionamiento como se dijo antes es sencillo y se basa en comparar lo que se encuentra al inicio del buffer y lo que está en la ventana. En caso de que se encuentre una coincidencia, se debe guardar la posición en la que esta se encuentra para así mirar luego si hay más caracteres que coincidan de forma continua. Dichas coincidencias marcarán la cantidad de posiciones que se desplacen las ventanas deslizables,lo cual ocurrirá hasta llegar al final de la cadena.

Para el inicio del algoritmo se llena el buffer de tamaño B, con B caracteres de la cadena; luego,se inicia la comparación de el buffer con la ventana tomando la cabecera del buffer (el carácter más a la izquierda), y se revisa si este ya se encuentra en la ventana, para ello, pueden existir 2 casos expuestos a continuación:

En primer lugar se tiene el caso en que el carácter no se encuentre dentro de la ventana, de ser así, las listas se desplazarán una posición a la derecha, dando como resultado que la cabecera del buffer pase ahora a la posición 1 de la ventana. El resultado para esta iteración estará dado por <0,0,X> siendo los 0 el indicativo que no se encontró una coincidencia y por lo tanto el largo de esta es de 0, y X el carácter que no se encontró en

la ventana y estaba de cabecera en el Buffer. Es importante recordar que la terna nos representa la posición de coincidencia, el largo de la coincidencia y carácter que termina en la posición uno de la ventana.

El segundo caso mencionado anteriormente es aquel en que la cabecera del buffer ya se encuentre en la ventana, el resultado de esta iteración viene dado por la posición en que se encuentre la coincidencia (más adelante se verá qué pasa si existe más de una coincidencia), para el largo de la coincidencia se empieza a mirar los caracteres que les siguen tanto a ambos (a la cabecera del buffer inicialmente y a la posición de la ventana donde se encuentra ese carácter), e ir acumulando estos hasta que se acabe la ventana o el buffer o hasta que no haya más coincidencia. El desplazamiento vendrá dado por la longitud de la coincidencia más uno, y en la última posición por el carácter que no coincidió que será igual que el anterior caso la posición 1 en la ventana.

Puede que en el proceso de codificación se den algunos casos especiales, estos se relacionan a continuación.

El primer caso especial que se encuentra es donde exista más de una coincidencia, por ejemplo tener en la cabecera del buffer una letra P y en la ventana tener dos P, una en la posición inicial y la otra en la final. Al no encontrar un consenso respecto al tema, se decidió plantear que se debe tomar la coincidencia más a la izquierda arbitrariamente, esto dado por la sencilla razón de que hay muchas más posibilidades de encontrar coincidencia, y en caso de que ambas tengan coincidencia, es mucho más probable que esta coincidencia tenga una longitud mayor.

El segundo caso especial que se puede encontrar es cuando todo el Buffer se encuentre dentro en la ventana, es decir que todos los caracteres del Buffer estén en el mismo orden dentro de la ventana, en este caso el Buffer se salta una iteración y trae el carácter que viene de la oración como carácter de ruptura. Cabe recalcar que en este caso se presenta la máxima longitud y al sumarle uno para realizar el desplazamiento, la lista de ventana llama al carácter que entraría a Buffer a entrar directamente a sí misma.

Para finalizar, los autores sugieren que al final de la cadena, cuando se termine de recorrer esta, en la terna final se coloque en la tercera posición algo que indique que ya se ha terminado, por lo que será <P,L,*> donde el * puede ser cambiado ya sea por una cadena o algo similar que indique que el proceso de codificación ha terminado.

Ya habiendo visto el funcionamiento podemos proseguir a ver cómo podemos decodificar este.

- Decodificación

El proceso de decodificación con las ternas generadas por el algoritmo LZ77 es mucho más sencillo que la misma codificación, y es que, para tal fin no se tiene en cuenta el buffer con el que se codificó, simplemente con los valores almacenados en las ternas se puede conocer la palabra que se desea encontrar.

De esta manera, para decodificar se empieza revisando el primer elemento de la terna, que como fue explicado en el ítem anterior se refiere a la posición de la coincidencia. De igual manera, es de resaltar que esta posición se debe leer del mismo modo en que fue escrita, esto se refiere a que si, por ejemplo, el algoritmo implementado se realizó con comprobaciones de derecha a izquierda, así mismo deben leerse estas posiciones.

Entonces, al rescatar el elemento mencionado, se procede a revisar su valor. Para ello, se tienen los siguientes casos dependiendo del dígito que se encuentre:

- Si el primer ítem de la terna es cero (0): Esto quiere decir que no se encuentran coincidencias antes de la terna actual, de modo que a la palabra decodificada simplemente se le suma la letra “siguiente” que reposa en el tercer ítem de la terna.

- Si el primer ítem de la terna es distinto de cero: Este caso implica que se encontró una coincidencia, por lo cual, el algoritmo de decodificación procede a calcular la posición “real” de dicha coincidencia (esto si se codificó de derecha a izquierda), y teniendo en cuenta este valor y la longitud de la coincidencia (que reposa en la segunda posición de la terna), se toma una copia de la subcadena que compone la coincidencia y se concatena, la palabra decodificada actual más la subcadena

mencionada más la letra “siguiente” que se encuentra en la tercer posicion de la terna.

Este proceso se repite revisando cada una de las ternas dadas para la decodificación. Así, al final de las ternas, cuando se detecte la cadena marcada como final de la decodificación, pueden darse dos casos:

- Si el primer ítem de la última terna es cero: Esto quiere decir que no hay más caracteres ni coincidencias, por lo cual la palabra decodificada hasta el momento es la definitiva, y no debe agregarse nada más a la misma.

- Si el primer ítem de la última terna NO es cero: Esto implica que se encontró una coincidencia, más sin embargo después de dicha coincidencia no se encuentra más información que decodificar, de modo que se agrega dicha coincidencia a la cadena y esta se da por finalizada.

Para la decodificación es mucho más sencillo el proceso y es que, en este caso solo nos interesa la ventana que, para facilidad la podemos ver de derecha a izquierda de manera creciente, y se cogerá el conjunto de ternas resultante de la codificación y empezamos a escribir estos.

VI. EJEMPLOS

Para ilustrar el funcionamiento del algoritmo y del aplicativo desarrollado en Java, a continuación se verán dos ejemplos con su respectiva codificación y decodificación, asumiendo que la cadena que indica el final del algoritmo es “EOF”.

Para nuestro segundo ejemplo se manejará la frase:

- Ejemplo 1: La oración dada es ‘CARLA LAVA LA TAZA’.

Inicialmente para codificar este ejemplo se manejó una ventana de tamaño 6 y un buffer de longitud 4.

Para empezar, se inicializa el buffer y se muestran las posiciones de la ventana como se había descrito anteriormente.

Ventana						Buffer							
						C	A	R	L	A		L	A	...
6	5	4	3	2	1									

Figura 2. Ventana, buffer y cadena del ejemplo 1, al inicio del algoritmo.

Con esto en mente, ya es posible realizar la primera iteración, obteniendo como resultado:

Ventana						Buffer								
					C	A	R	L	A		L	A	V	...

Figura 3. Ventana, buffer y cadena del ejemplo 1, luego de la primera iteración.

Como se evidencia en la imágen anterior, la cabecera del Buffer quedó en la posición 1 de la ventana, y la terna resultante fue <0,0,C>, las siguientes 3 iteraciones serán repeticiones sin coincidencia por lo cual, para efectos de avanzar en la explicación, se saltará a la muestra gráfica de dichas iteraciones, dando como resultado: :

Ventana						Buffer								
		C	A	R	L	A		L	A	V	A		L	...

Figura 4. Ventana, buffer y cadena del ejemplo 1, en su cuarta iteración.

Luego, la cadena de salida actual iría con <0,0,C>, <0,0,A>, <0,0,R>, <0,0,L>. A continuación se encuentra la primera coincidencia, dándose esta en la posición 3 y además evidenciándose que es el único lugar donde se encuentra una coincidencia. Es evidente que la cadena de coincidencia sólo coincide con un carácter, por lo cual el desplazamiento será de dos (es de recordar que se desplaza el largo de la coincidencia más 1). Así, La terna resultante será <3,1,” ”>, para continuar se reescribe la tabla de la siguiente manera:

Ventana						Buffer								
C	A	R	L	A		L	A	V	A		L	A		...

Figura 5. Ventana, buffer y cadena del ejemplo 1, en su quinta iteración.

Para la siguiente iteración, se evidencia una segunda coincidencia y es posible ver a simple vista que coinciden 2 caracteres (LA), por lo cual se hará un desplazamiento de 3, y la terna resultante de ello será <3,2,V>. De esta manera, la cadena de ternas actual está dada por: <0,0,C>, <0,0,A>, <0,0,R>, <0,0,L>, <3,1, “ ”>, <3,2,V>.

Reescribiendo la tabla se tiene que:

Ventana						Buffer								
L	A		L	A	V	A		L	A		T	A	Z	A

Figura 6. Ventana, buffer y cadena del ejemplo 1, en su sexta iteración.

En la siguiente iteración se denotan los 2 casos especiales que se describieron anteriormente, siendo el primero que se encuentran 2 coincidencias, sin embargo, como se dijo anteriormente, se toma la de mayor posición en la ventana, esto basado en un análisis de favorabilidad, puesto que la coincidencia es de todo el buffer, por lo cual la tercera parte de la terna vendrá dada por el carácter de la cadena que viene en camino, para este caso al evidenciar que los 4 caracteres coinciden, se tiene la siguiente terna <5,4, ‘ ’>.

De esta manera, la reescritura de la tabla está dada por:

Ventana						Buffer				
V	A		L	A		T	A	Z	A	

Figura 7. Ventana, buffer y cadena del ejemplo 1, en su séptima iteración.

Así, se evidencia que la T con ningún carácter dentro de la ventana, por lo que se genera la terna <0,0,T >, la cual se añade a la cadena de ternas (nuestra salida) que va en <0,0,C>, <0,0,A>, <0,0,R>, <0,0,L>, <3,1, “ ”>, <3,2,V>, <5,4, “ ”>, <0,0,T>. para proseguir, se tiene la tabla:

Ventana						Buffer				
A		L	A		T	A	Z	A		

Figura 8. Ventana, buffer y cadena del ejemplo 1, en su octava iteración.

De esta manera, es posible ver que ya no queda más texto de la cadena, continuando se presenta una coincidencia en la posición 6 de la ventana. Así, se tendrá la terna <6,1,Z> para esta iteración, luego se realiza el respectivo desplazamiento de 2 por lo cual queda un Buffer casi vacío, que se presenta a continuación en la ventana:

Ventana						Buffer			
L	A		T	A	Z	A			

Figura 9. Ventana, buffer y cadena del ejemplo 1, en su última iteración.

Finalmente, se evidencia una coincidencia en 5, y es posible evidenciar que no hay más caracteres para llevar a la ventana por lo cual es donde aparece la cadena de finalización definida anteriormente (“EOF”), por lo tanto, se obtiene el siguiente resultado representado en la tabla expuesta a continuación:

<0,0,C>
<0,0,A>
<0,0,R>
<0,0,L>
<3,1,"">
<3,2,V>
<5,4," ">
<0,0,T>
<6,1,Z>
<5,1,EOF>

Figura 10. Tabla con las ternas dadas como resultado de la codificación del primer ejemplo.

Para la decodificación, se toma la anterior tabla iniciando desde arriba hacia abajo, de modo que, aplicando lo expuesto teóricamente en esta investigación, dado que las primeras 4 ternas, el primer carácter es un cero (0), estas se van agregando al texto decodificado, siendo como resultado parcial hasta la iteración 4 la palabra:

CARL

Para la siguiente terna, como el primer carácter es 3 y la longitud de la coincidencia es de 1, se deben contar las posiciones iniciando de derecha a izquierda, de modo que la subcadena de coincidencia está dada por “A”, así, el nuevo resultado parcial estára dado por concatenar lo que se lleva acumulado más la subcadena descrita anteriormente más la posición tres de la terna actual, que describe la letra “siguiente”, dando como salida:

CARLA_

Donde el carácter “_” representa un espacio vacío. Luego, para la siguiente iteración, como la posición de la coincidencia es 3 y la longitud es 2 con el siguiente carácter como “v”, la subcadena de coincidencia estará dada por “LA” y el resultado parcial nuevo será:

CARLA_LAV

Así, para la séptima iteración, se tiene la subcadena de coincidencia “A LA” y el resultado parcial estará dado por:

CARLA_LAVA_LA_

Así, en la siguiente iteración, dado que no se encuentra coincidencia, el resultado actual se concatena con el “siguiente” carácter y se avanza a la penúltima iteración, donde se encuentra una coincidencia en la posición 6 de la ventana con

longitud de 1 y siguiente carácter de “Z”, dando como resultado la cadena:

CARLA_LAVA_LA_TAZ

Finalmente, como se evidencia que la “letra siguiente” final está dada por EOF y se encontró una coincidencia en la posición 6 de la ventana con longitud de 1 (“A”), la iteración final da como resultado la frase:

CARLA_LAVA_LA_TAZA

Y la cadena “EOF” marca que no se encuentran más ternas, por lo cual, es posible afirmar que el proceso de decodificación ha finalizado.

- Ejemplo 2: La oración dada es “ENSALADA SALADA”.

Ahora se verá el segundo ejemplo en el cual se manejara una ventana de tamaño 6 y un buffer de longitud 3.

Inicialmente, como en el ejemplo anterior se llena el Buffer y se dibuja la ventana de la siguiente manera:

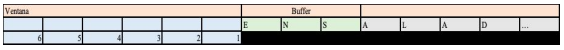

Figura 11.. Ventana, buffer y cadena del ejemplo 2, al inicio del algoritmo.

Se ve que se pueden hacer por lo menos 5 iteraciones y aun no se encontrará coincidencia, por lo cual se puede tener la siguiente tabla luego de las primeras iteraciones:

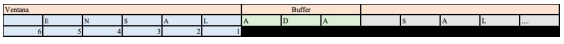

Figura 12. Ventana, buffer y cadena del ejemplo 2, en su sexta iteración.

Se evidencia la primera coincidencia con un largo de 1, en la posición 3, por lo cual al actualizar el listado de ternas se encuentra este por ahora de la siguiente manera: <0,0,E>, <0,0,N>, <0,0,S>, <0,0,A>, <0,0,L>, <2,1,D>.

De tal forma que, a continuación se tiene como resultado la ventana, el buffer y la cadena:

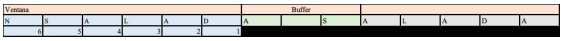

Figura 13. Ventana, buffer y cadena del ejemplo 2, en su séptima iteración.

Donde es posible evidenciar la coincidencia en 4 con un largo de 1, por lo cual respecto a la terna se da como resultado: <4,1, ‘ ’ > y se tiene la siguiente tabla:

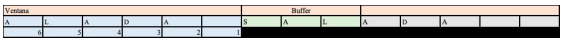

Figura 14. Ventana, buffer y cadena del ejemplo 2, en su octava iteración.

De nuevo se encuentra con que no hay coincidencia pero se ve que luego de desplazarse, el siguiente carácter coincidirá en la posición 5 con un largo de 1, por lo cual estas 2 iteraciones generan las ternas <0,0,S>, <5,1,L> y así la nueva tabla estará dada por:

Figura 15. Ventana, buffer y cadena del ejemplo 2, en su décima iteración.

Aquí se evidencia que la frase se ha terminado por lo cual el buffer no recibirá nada más, para terminar se realizan las 2 iteraciones finales y se obtienen las ternas <5,1,D>, <4,1,EOF>

Finalmente el resultado codificado se tiene en la siguiente tabla:

<0,0,E>
<0,0,N>
<0,0,S>
<0,0,A>
<0,0,L>
<2,1,D>
<4,1,‘ ’>
<0,0,S>
<5,1,L>
<5,1,D>
<4,1,EOF>

Figura 16. Tabla con las ternas dadas como resultado de la codificación del segundo ejemplo.

Respecto a la decodificación, a manera de resumen se tiene la siguiente tabla que ilustra cómo funciona el algoritmo de decodificación en cada una de las iteraciones:

Tripleta	Subcadena	Res. Parcial
<0,0,E>	//	E
<0,0,N>	//	EN
<0,0,S>	//	ENS

<0,0,A>	//	ENSA
<0,0,L>	//	ENSAL
<2,1,D>	A	ENSALAD
<4,1,"_">	A	ENSALADA _
<0,0,S>	//	ENSALADA _S
<5,1,L>	A	ENSALADA _SAL
<5,1,D>	A	ENSALADA _SALAD
<5,1,EOF>	A	ENSALADA _SALADA

Tabla 1. Ilustración de los datos más importantes en cada iteración para la decodificación..

VII. IMPLEMENTACIÓN

Posterior a comprender de una manera clara y concisa el funcionamiento del algoritmo LZ77, se decidió realizar su implementación empleando el lenguaje de programación Java, siendo como resultado un aplicativo que es capaz de codificar y decodificar texto basado en el algoritmo descrito anteriormente.

De esta manera, se ilustra a continuación la interfaz del programa realizado:

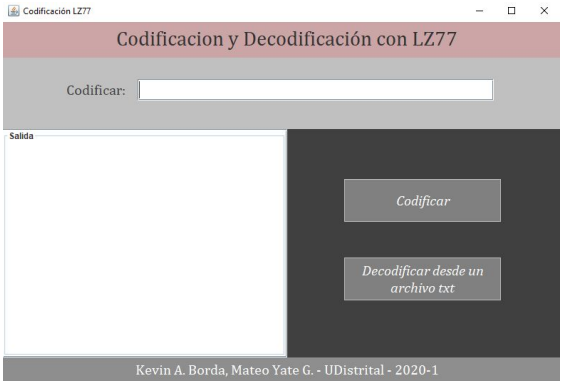

Figura 16. Ventana principal del aplicativo desarrollado.

Allí se cuenta con 5 paneles (o subdivisiones) en la GUI del programa, la de la parte superior ilustra el título del aplicativo, la siguiente define un campo de texto para recibir la frase o caracteres que el usuario decodificar, luego, en la parte izquierda se encuentra un panel con un área para ilustrar la salida, bien sea de codificación o de decodificación, a la derecha de este panel se encuentran los botones para realizar cada acción de algoritmo, y finalmente en la parte inferior del aplicativo reposa la información de los desarrolladores del programa.

Entonces, se demostrará el funcionamiento del aplicativo por medio de los ejemplos expuestos anteriormente.

Para el primer ejemplo, si se tiene la cadena 'CARLA LAVA LA TAZA' con una ventana de 6 y un buffer de 4, se procede a codificar esta frase de la siguiente manera luego de pedir dichos valor y verificar que todo se encuentre en orden respecto a la escritura y los tamaños:

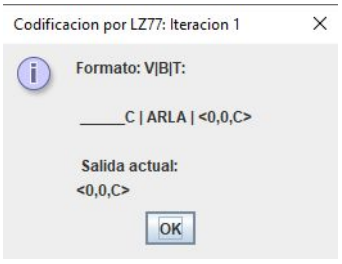

Figura 17. Primera iteración de codificación del texto "CARLA LAVA LA TAZA" en el aplicativo desarrollado.

Así, para cada iteración se despliega una ventana emergente que muestra el número de iteración actual, el formato en que se muestran los datos, la ventana actual, el buffer actual y la tripleta generada, y la salida acumulada del programa hasta el momento.

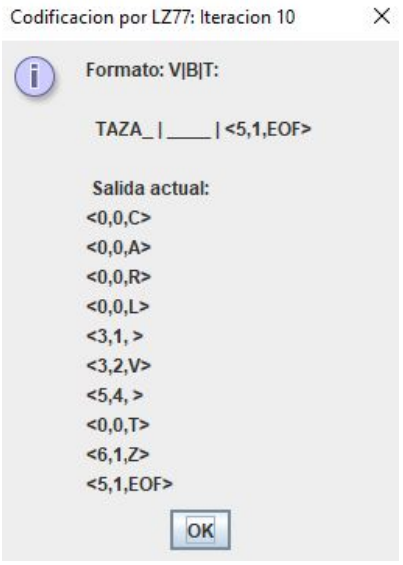

Figura 18. Última iteración de codificación del texto "CARLA LAVA LA TAZA" en el aplicativo desarrollado.

A manera de ejemplo, esta es la iteración final para el ejemplo en cuestión, donde se denota gracias al EOF y al buffer vacío que el programa finalizó el proceso de codificación, luego de ello, el programa exporta los resultados en forma de ternas a un archivo de texto plano para así poder almacenar cada proceso que se lleve a cabo.

Luego, para el proceso de decodificación, se procede a realizar el cargue de un archivo de texto plano con un formato como el que se presenta en el archivo de codificación generado, con una serie de ternas divididas por renglones sin ningún caracter o espacio por fuera de estas.

De esta forma, la ventana inicial si se toma como archivo de texto el archivo generado en la codificación para el presente ejemplo, será:

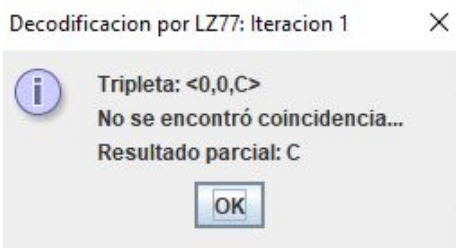

Figura 19. Primera iteración de decodificación del texto "CARLA LAVA LA TAZA" en el aplicativo desarrollado.

Es importante recalcar que para cada iteración la información desplegada varía dependiendo si existe o no una coincidencia correspondiente a la tripleta o terna actual. Para el caso actual, como no se encontró coincidencia, no se despliega información adicional a la suministrada en la ventana expuesta anteriormente.

A manera de ejemplo, si se encuentran coincidencias, se presenta la siguiente ventana:

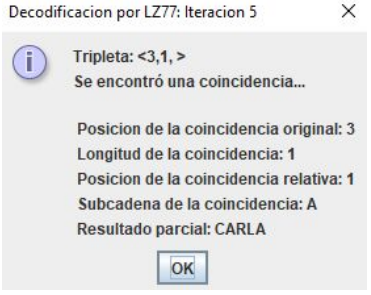

Figura 20. Quinta iteración de decodificación del texto "CARLA LAVA LA TAZA" en el aplicativo desarrollado.

Donde, como es evidente, se explican los datos iniciales de la tripleta y la posición relativa que se refiere a la posición leída de derecha a izquierda en el resultado parcial, así como el acumulado actual.

Finalmente, se realiza el despliegue de una ventana emergente que indica el resultado final de decodificación, que para el presente ejemplo será:

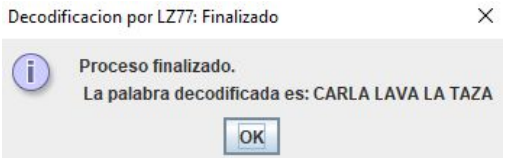

Figura 21. Resultado final de decodificación del texto "CARLA LAVA LA TAZA" en el aplicativo desarrollado.

Y se imprimen los datos en el área correspondiente de la ventana principal del programa.

De igual manera, como cierre del actual documento investigativo, se realizó el segundo ejemplo expuesto en la sección "EJEMPLOS" del presente documento, dando como resultado para el proceso de codificación lo siguiente:

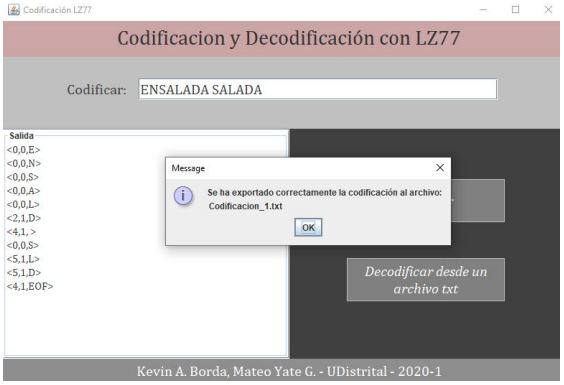

Figura 22. Resultado del proceso de codificación del texto "ENSALADA SALADA" en el aplicativo desarrollado.

Y para el proceso de decodificación con base en el archivo generado, lo expuesto a continuación:

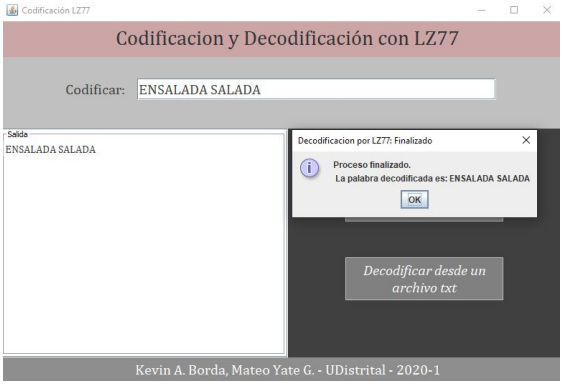

Figura 23. Resultado del proceso de decodificación del texto "ENSALADA SALADA" en el aplicativo desarrollado.

Finalmente, el proyecto se encuentra alojado en un repositorio de GitHub de autoría propia para la consulta de quien así lo desee. [7]

VIII. BIBLIOGRAFÍA

[1] 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

[2] 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

[3] 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

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

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

[6] 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

[7] M. Yate & K. Borda, "mateoyateg/CodificadorLZ77", [En línea]. *GitHub.com*, Disponible en: https://github.com/mateoyateg/CodificadorLZ77.