Compresión basada en LZ77 y variante FastQ

Tarea 1
Tutor: Álvaro Martin

Alexis Baladón Ferreira de Araújo, 5574612-4

Compresión de Datos sin Pérdida 2021
Núcleo de Teoría de la Información
Facultad de Ingeniería. Universidad de la República
Montevideo, Uruguay

Abstract

En este informe se mostrará el proceso de implementación de un programa capaz de comprimir tanto como descomprimir archivos en base al algoritmo LZ77, además de una extensión del mismo que optimiza la compresión de archivos de formato FastQ. A su vez se analizará su eficiencia en términos de capacidad de compresión, comparándola con la de archivos Zip, tambien basados en LZ77. Finalmente se observan resultados más óptimos en la variación para archivos FastQ, aunque menos que en la realizada por Zip.

1 Introducción

LZ77 es un algoritmo de compresión sin pérdida propuesto en el año 1977 por Abrahm Lempel y Jacob Ziv, el cual es capaz de codificar una secuencia de forma óptima y universal. Este tipo de métodos de compresión ha sido muy popular a lo largo de la historia. Actualmente, muchos de los compresores más populares (como Zip o gzip) se basan en variaciones del mismo.

Al utilizar este algoritmo se busca biseccionar la información a comprimir en: los últimos símbolos comprimidos, y los que aun no lo han sido. De aquí se intentará expresar la mas reciente información comprimida en función a la que aun no lo ha sido, para luego repetir el proceso.

2 Problema a Resolver

El objetivo de este informe es llevar a cabo y detallar la implementación de un programa capaz de comprimir tanto como descomprimir archivos aprovechándose del algoritmo LZ77, el cuál recibirá como parámetro un número $N \in \mathbb{N}$ que determinará tanto el largo de la ventana corrediza (detallada en [2]) como el largo máximo de la información construida a partir de esta en cada iteración.

Luego de su creación se buscará adaptar el programa a archivos de texto FormatQ, los cuales siguen un formato característico que facilita su compresión con este tipo de algoritmos [1].

3 Metodología

3.1 Compresor y descompresor genéricos

El primer objetivo del programa era ser capaz de comprimir tanto como descomprimir cualquier tipo de archivo de texto con la ayuda del algoritmo LZ77. Para esto fue necesario desarrollar un esquema que permitiera visualizar con facilidad las etapas de compresión del archivo y el uso de memoria que cada fase requeriría.

Para segmentar esta información se visualizaron las siguientes etapas:

1) Conocer el alfabeto del archivo a manipular.
2) Comprimir sus primeros N símbolos.
3) Comprimir las coincidencias $(largo, offset)$ o caracteres en caso de no haberlas.

Primera etapa:
La primera etapa no es realmente necesaria, ya que podemos suponer que cada símbolo leido por la consola pertenece un caracter entre 0 y 255 (según la tabla ASCII). Sin embargo no es frecuente que un archivo contenga todos los símbolos del alfabeto al que pertenecen (por ejemplo, no es normal que un archivo de texto contenga más de los 95 Caracteres Imprimibles), por lo que se realizará una asignación de un código de tamaño fijo ($\lceil log|A| \rceil$) a cada símbolo observado. Por lo tanto si se utilizan no más de 128 caracteres se podrá afirmar que esta característica ahorrará al menos $n/8$ bits en la compresión final, siendo n la cantidad de símbolos en el archivo original.

Para comprimir esta etapa, se comprime en primer lugar el largo del alfabeto al cual llamaremos $|A|$, que al estar acotado por 256 podrá expresarse usando 8 bits. A partir de aquí se imprimirán los códigos de $|A|$ caracteres, lo cual requerirá el uso de $|A| * \lceil log|A| \rceil$ bits. No será necesario especificar su codificación en el archivo, el descompresor sabrá esto según el orden en el que fueron impresos los caracteres (el primer caracter se codifica como 0, el segundo como 1, el tercero como 2, etc).

En caso de que el tamaño del alfabeto sea mayor que 128 no tendría sentido el uso de esta funcionalidad. Si ese es el caso se imprimirá el bit '0' al inicio del archivo comprimido. En caso contrario se imprimirá un 1.

Segunda etapa: En esta etapa se hace uso de la información obtenida en la etapa anterior. Imprimiéndose los N caracteres según su nueva asignación de código. Esta etapa ocupará $N * \lceil log|A| \rceil$ bits de memoria.

Tercera etapa:
Para la tupla $(largo, offset)$ se asignará un código a cada número de tamaño fijo (Imprimiendo los $\lceil log|N| \rceil$ bits menos significativos de los mismos). Esto es posible ya que ambos números enteros son positivos y no se permitirá que superen a N como tamaño. Cada tupla requerirá de $2 * \lceil log|N| \rceil$ bits.

Para los caracteres simplemente se codificará según la asignación que se le dio en la primera etapa, que como fue descrito en la misma ocuparán $\lceil log|A| \rceil$ bits.
Además, para saber si una frase es coincidencia o no se imprimirá un 1 o un 0 respectivamente previo a su aparición.

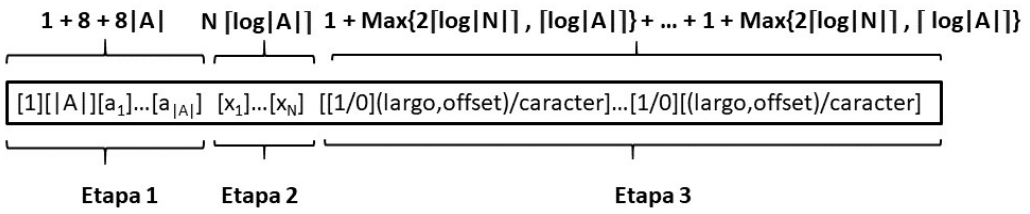

Fig. 1. Representación gráfica y cota de compresión

3.2 Compresor y descompresor FastQ

Como ya fue mencionado, los archivos FastQ tienen un formato característico que permitirá adecuar el programa para obtener mejores resultados en cuanto a memoria. Cada archivo de texto puede dividirse en conjuntos de cuatro lineas (esto es posible ya que todo archivo FastQ tiene un número de filas múltiplo de cuatro), de las cuales interesarán las siguientes propiedades:

1) La segunda fila tiene siempre el mismo número de caracteres que la cuarta
2) Cada fila tiene un alfabeto acotado y conocido, en particular el alfabeto de la primera y tercera fila coinciden.

3) Al inicio de las primeras filas siempre aparece el símbolo '@', y en las terceras '+'.

Esta información permitirá particionar el archivo para luego comprimir cada segmento por separado con una mayor tasa de compresión, además de poder obviar los caracteres de la propiedad 3. Los segmentos comprimidos por separado serán:

1) Los largos de las segundas y cuartas filas

En un principio el largo los reads no está acotado superiormente, por lo que se decidió utilizar la codificación para enteros positivos "Elias-gamma", el cual representa al número 'x' escribiendo $\lfloor logx \rfloor$ ceros, seguidos de un '1' y los $\lfloor logx \rfloor$ bits menos significativos del mismo. Dada la tendencia de los archivos FastQ de usar largos de lineas similares, se determinó comprimir la cantidad de veces que se repetía el largo 'l' seguido del valor del mismo (utilizando la misma codificación). Para determinar si se debe seguir leyendo largos se imprime un '1' antes de cada uno o un '0' en caso de ingresar al siguiente segmento. Llamemos $Elias(w)$ a la función que codifica a una secuencia de números w, entonces por ejemplo la compresión de los largos 72-72-64-72 sería: 1 Elias(2) Elias(72) 1 Elias(1) Elias (64) 1 Elias(1) Elias(72) 0.

En el peor de los casos este segmento imprime $2 * \lfloor log1 \rfloor + 1 + 2 * \lfloor logx \rfloor + 1$ bits por largo, siendo 'x' el largo comprimido (caso en el que todos los largos son distintos).

2) Las segundas filas

Al conocer sus largos será posible comprimir cada segmento excluyéndose los fines de linea. Su compresión será equivalente a la genérica sin necesidad de atravesar la nombrada 'primera etapa' (3.1) . Notar que el alfabeto de estas filas tiene orden 5.

3) Las cuartas filas

Se aplicará el mismo procedimiento que a las segundas filas. Notar que el alfabeto de estas filas tiene orden 94.

4) Las primeras y terceras filas

Aquí el largo largo del alfabeto es conocido, sin embargo se decidió hacer un recorrido por el mismo con el fin de acotar la representación de sus símbolos de la misma forma que en la 'primera etapa' de la compresión genérica (3.1).

Al igual que en la compresión para archivos genéricos, se mostrará cada segmento (en el mismo orden que son comprimidos) de forma gráfica con el fin de facilitar su comprensión.

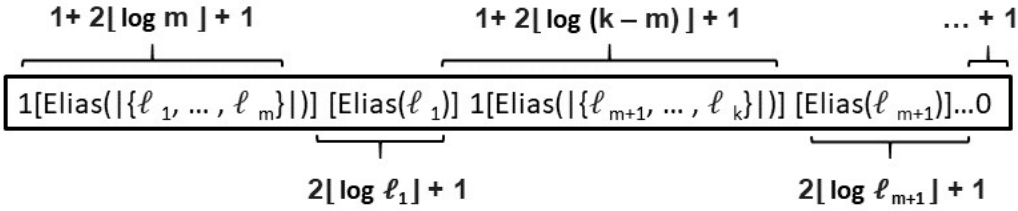

Fig. 2. Representación gráfica Segmento 3

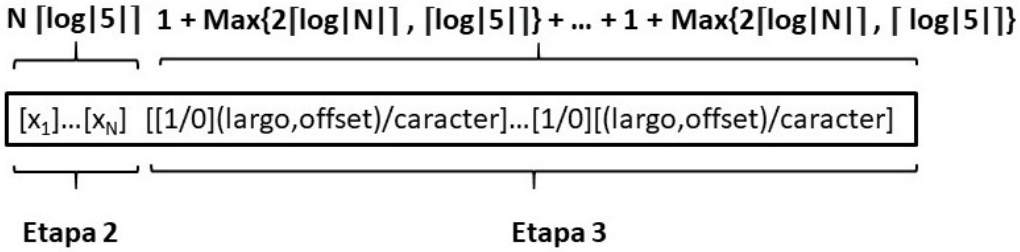

Fig. 3. Representación gráfica Segmento 1

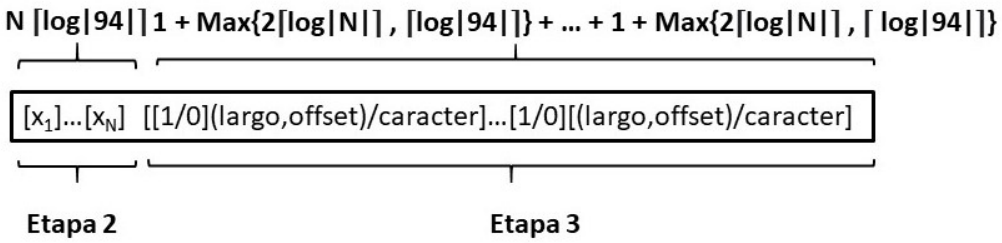

Fig. 4. Representación gráfica Segmento 2

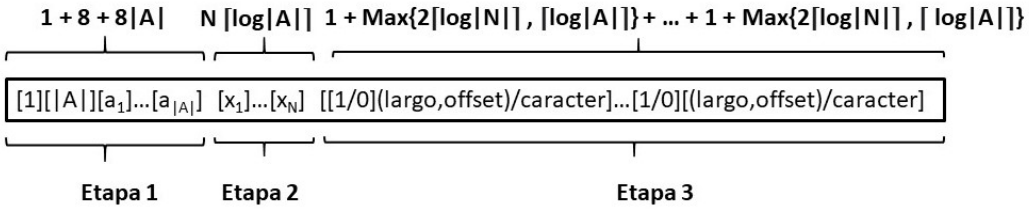

Fig. 5. Representación gráfica Segmento 4

4 Análisis Experimental

4.1 Compresor y descompresor genéricos

Los experimentos realizados sobre este apartado están centrados en la tasa de compresión del programa variando el largo del parámetro N ([2]), donde se utiliza en todo momento un N potencia de dos, con el fin de minimizar la cantidad de bits ocupados en las codificaciones de tamaño $\lceil log|N| \rceil$ bits. Para esto fueron analizados distintos archivos de los cuales se obtuvieron los resultados expuestos a continuación.

A tener en cuenta: Para esta sección se utiliza el término 'tasa de compresión' para referirse al cociente de la cantidad de caracteres comprimidos y el tamaño del archivo original.

4.1.1 Texto experimental: "Moby Dick" (1.3MB)

Para este archivo se vario el parámetro N acorde al tamaño original del mismo, observando su tasa de compresión a medida que se creaba el archivo comprimido. Los resultados obtenidos son los siguientes:

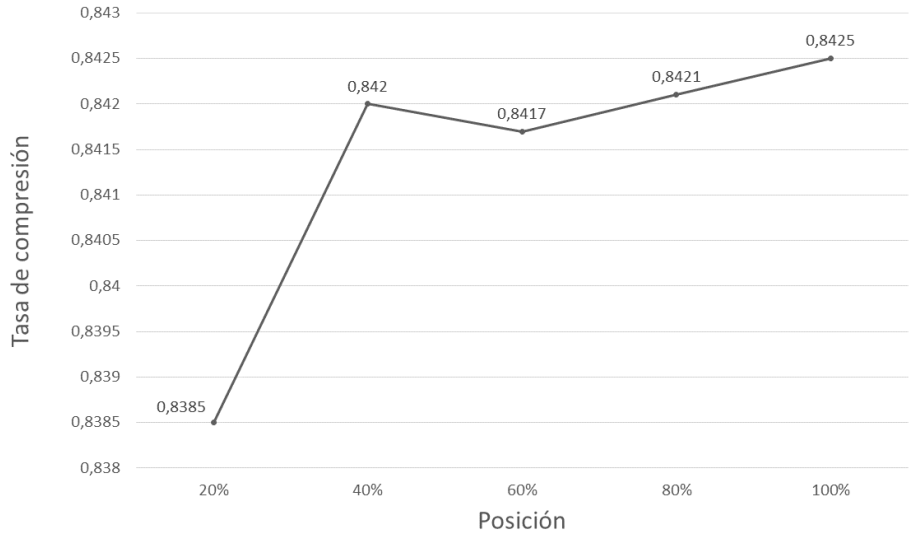

Fig. 6. Tasa de compresión en función a posición de *"Moby Dick"* para N fijo (512)

En esta gráfica (dejándose el valor N fijo en 512, aunque sin perderse generalidad), se observa como la tasa de compresión aumenta a medida que se comprime la secuencia, a excepción claro del segmento entre el 40 y 60%. Hasta el momento se pueden considerar las hipótesis de que el comportamiento esperado del programa es que la tasa crezca cada vez más lento a medida que la compresión avanza.

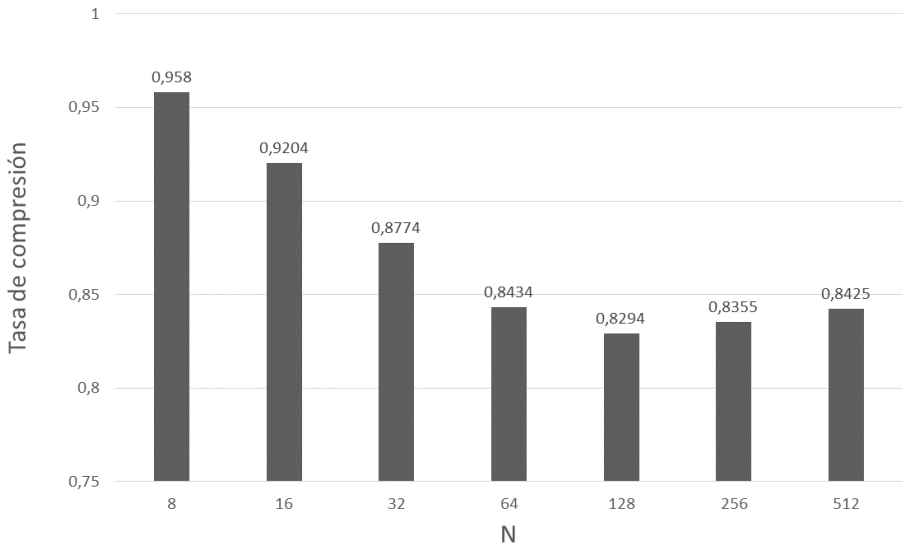

Fig. 7. Tasa de compresión en función de tamaño de la ventana de *"Moby Dick"*

Aquí se observa como la tasa de compresión disminuye hasta alcanzar un mínimo en $N = 64$. Analizando los N inmediatamente próximos a este da la impresión de que no habrá otro parámetro capaz de optimizar estos resultados. Sin embargo, utilizando el parámetro $N = 4096$ se pudo observar una tasa de compresión de 0.8045, la cual es aun menor a la obtenida con $N = 64$. Adicionalmente, al comprimir este mismo texto en un archivo Zip se logró una tasa de compresión de 0.414, tasa casi el doble de pequeña que la obtenida por el programa.

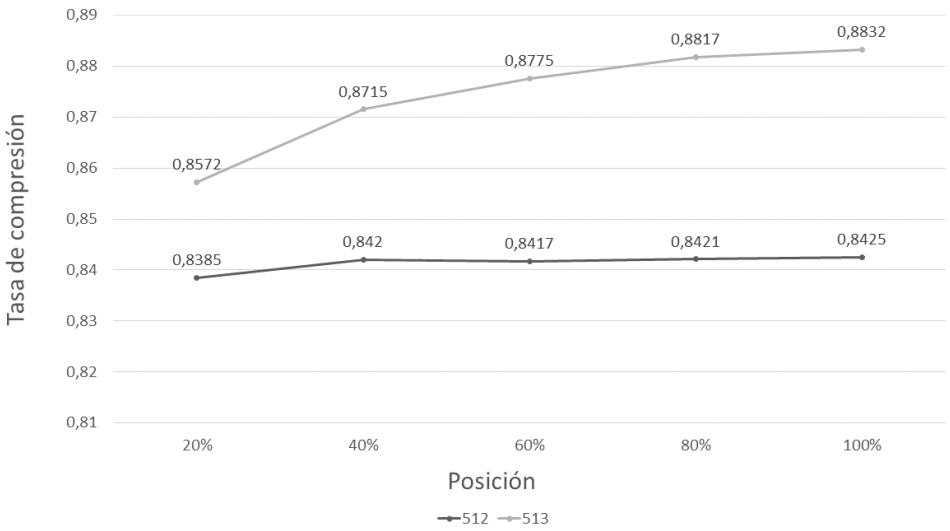

Fig. 8. Comparación N potencia de 2 de *"Moby Dick"*

Como es de observar, la tasa de compresión cuando N no es potencia de 2 es considerablemente mayor. La decisión de estudiar estos tamaños en particular ya había sido justificada, sin embargo ahora se obtienen resultados empíricos que refuerzan la decisión.

	20%	40%	60%	80%	100%
8	0,9584	0,9588	0,9583	0,9585	0,958
16	0,9222	0,9224	0,9215	0,9212	0,9204
32	0,8803	0,8793	0,8789	0,8785	0,8774
64	0,8465	0,8462	0,8453	0,8447	0,8434
128	0,8309	0,8318	0,8313	0,8304	0,8294
256	0,8343	0,8372	0,8368	0,8362	0,8355
512	0,8385	0,842	0,8417	0,8421	0,8425
513	0,8572	0,8715	0,8775	0,8817	0,8832
4096	0,794	0,8016	0,8016	0,80155	0,8045

Fig. 9. Datos finales de *"Moby Dick"*

Observarción: Los datos exhibidos son de la tasa de compresión **desde el inicio de la compresión hasta el momento indicado**, **no** de cada segmento por separado.

4.1.2 Texto experimental: "El señor de los anillos" (3.1MB)

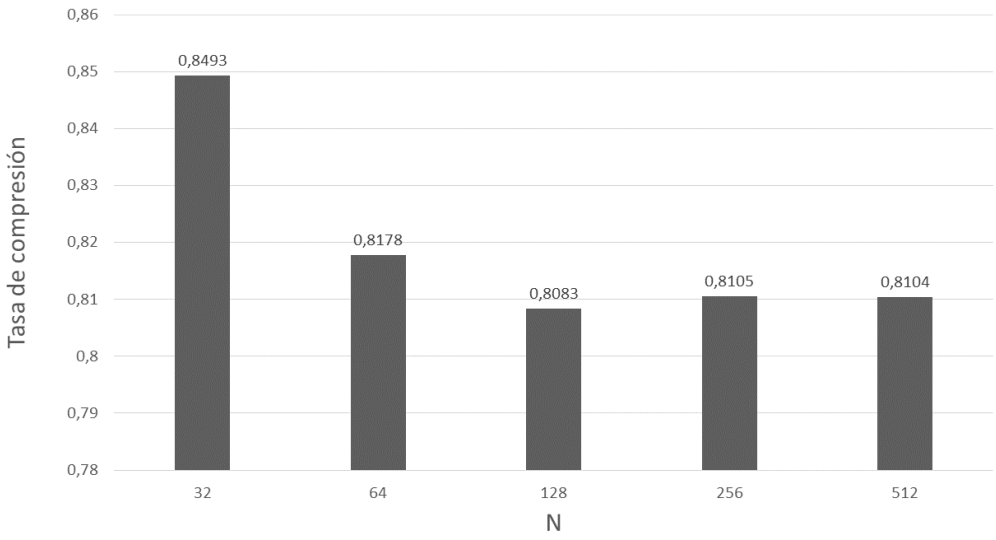

Fig. 10. Tasa de compresión en función de tamaño de ventana de *"El Señor de los Anillos"*

De aquí se observa un comportamiento similar al archivo anterior. Se da un valor que minimiza la tasa de compresión localmente, y aunque nuevamente se de la impresión de ser global, al analizar los resultados con $N = 4096$ se vuelve a lograr una tasa de 0.7566 aun menor que las anteriores.

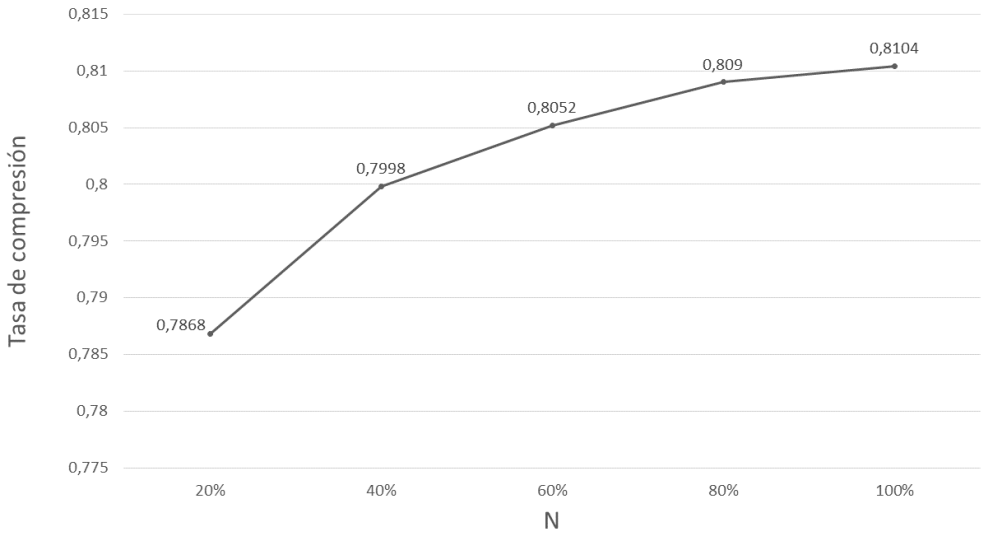

Fig. 11. Tasa de compresión en función a posición de *"El Señor de los Anillos"* para N fijo (512)

Como en el caso anterior, se dan comportamientos similares con respecto al crecimiento de la tasa a medida que la compresión avanza.

Adicionalmente, al intentar comprimir el texto comprimido en un archivo Zip, se obtuvo en general una mejor tasa de compresión que en el compresor LZ77, aunque nunca mejor que la compresión del archivo original en un .zip. Por ejemplo, la tasa de compresión para $N = 4096$ era 0.7566; la tasa del archivo comprimido comprimido por Zip es de 0.5678; mientras que la del archivo original comprimido solo con Zip es de 0.3781.

	20%	40%	60%	80%	100%
32	0,8221	0,8391	0,8391	0,845	0,8493
64	0,7912	0,8071	0,8131	0,8165	0,8178
128	0,7808	0,7969	0,803	0,8067	0,8083
256	0,7836	0,7988	0,8049	0,809	0,8105
512	0,7868	0,7998	0,8052	0,809	0,8104
513	0,8572	0,8715	0,8775	0,8817	0,8832
4096	0,7363	0,747	0,7505	0,7553	0,7566

Fig. 12. *Datos finales de "El Señor de los Anillos"*

Observación: Aunque no es mencionado se muestra la tasa de compresión con $N = 513$, lo cual reafirma lo anterioremente dicho sobre las potencias de 2.

4.1.3 Texto experimental: "Texto aleatorio (1.0MB)"

Es fácil notar que, al utilizar este algoritmo, la tasa de compresión mejora mientras más coincidencias y patrones el algoritmo es capaz de encontrar. En esta sección se analizará un archivo con un alfabeto de orden 64 generado de forma aleatoria, lo cual minimizaría el encuentro de patrones en el mismo. Este es generado gracias al siguiente comando de Linux:
base64 /dev/urandom — head -c 10000000 > aleatorio.txt

Los datos experimentales sobre el mismo fueron:

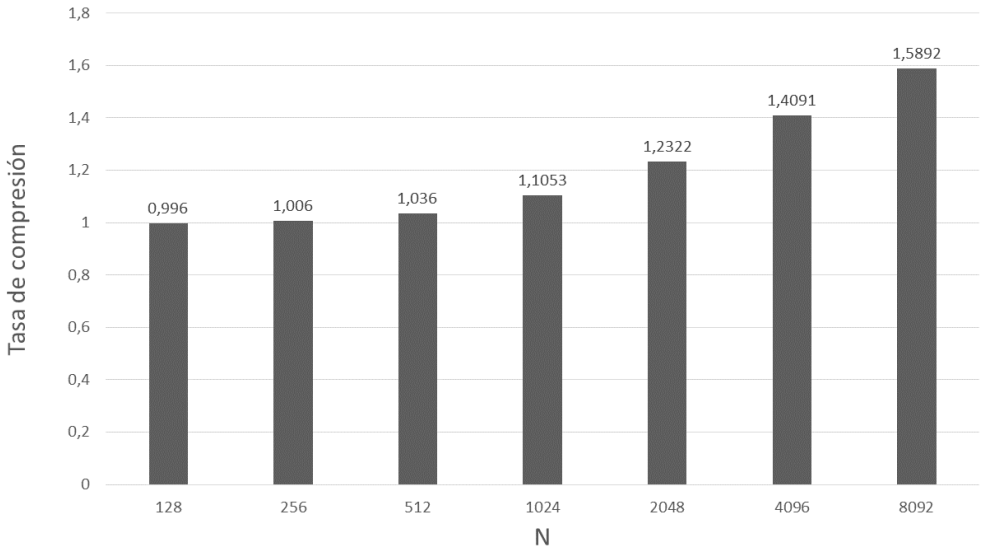

Fig. 13. Grafica *Tiempo de compresión en función de N*

Como es de observarse, aun con una ventana de gran tamaño, la tasa de compresión sobrepasa en todo caso a 1, lo cual podría ser catastrófico si se trabajase con archivos de gran porte. Además la gráfica es creciente, por que lo da la impresión de que fue encontrado un caso borde del algoritmo, que es capaz de refutar su optimalidad. Sin embargo, intentando con un N notablemente mayor, 65536, se obtiene una tasa de compresión de 0.86. Este fenómeno será discutido en la sección conclusiones.

4.2 Compresor y descompresor FastQ

En esta sección será analizada la compresión de archivos FastQ, comparándose a la realizada por el compresor de archivos genéricos.

4.2.1 Texto experimental: ”Archivo FastQ (14MB)”

Trabajando con este archivo se logró comprobar la efectividad de esta nueva implementación, consiguiéndose una tasa menor en todos los casos probados, a tal punto que la menor compresión lograda con la versión genérica del programa no alcanzó a la mayor de su adaptación a este tipo de archivos.

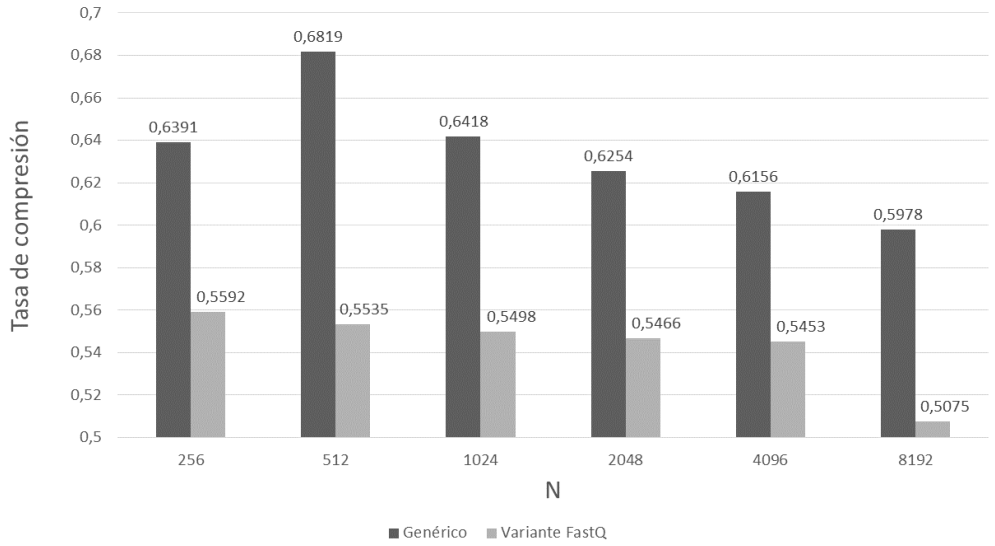

Fig. 14. Comparación de métodos genérico y adaptado con tasa de compresión en función de N (Archivo 14MB)

Observación: Notar que el eje vertical comienza con el valor 0.5, lo cual podría dar a entender que la diferencia entre ambos métodos es mayor de lo que realmente es.

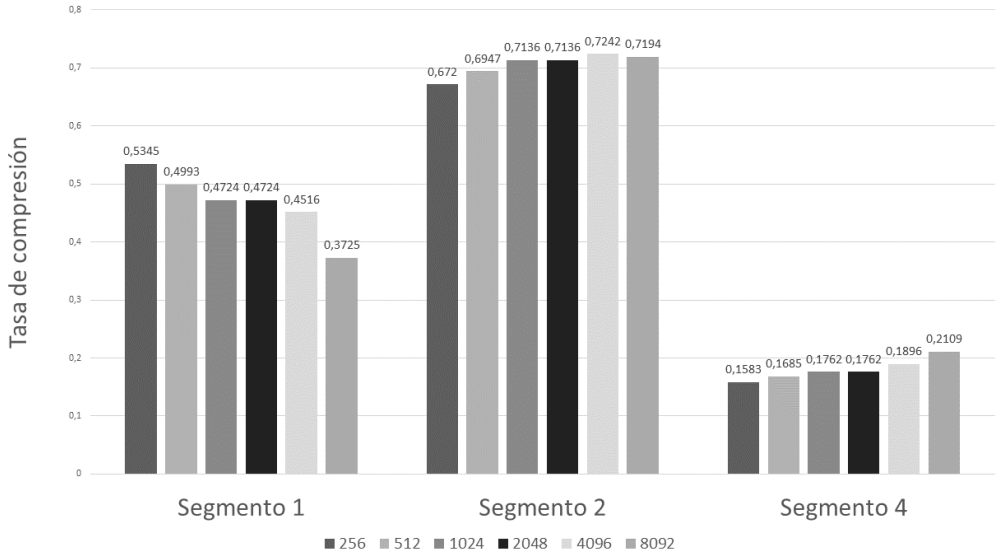

Fig. 15. Compresión según N para cada segmento (Archivo 14MB)

Aquí la diferencia entre la compresión de cada segmento es notoria. Parece ser que hay segmentos mucho más fáciles de comprimir que otros.

Comprimiendo este mismo archivo utilizando Zip se obtuvo una tasa de compresión de 0.3192, tasa menor a la obtenida para cualquiera de los valores de N experimentados.

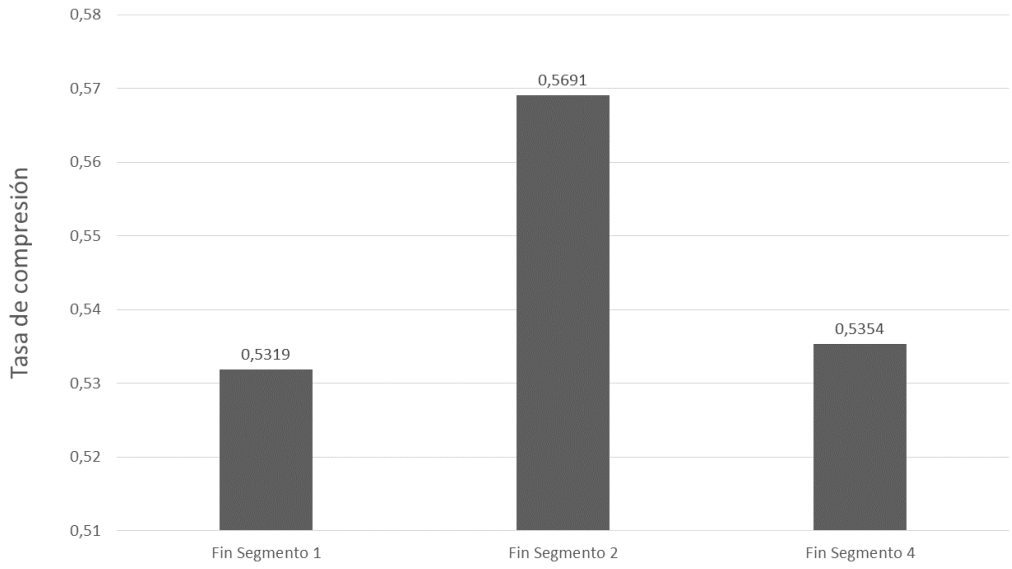

Fig. 16. Tasa de compresión a medida que avanzan segmentos (Archivo 14MB, $N = 64$)

Con respecto a cómo avanza la tasa de compresión, era de esperarse que en los momentos que la tasa en el siguiente segmento es menor, la tasa comience a disminuir, mientras que en el caso contrario aumente de forma monótona (ya que son compresiones independientes del algoritmo original sobre cada segmento).

4.2.2 Texto experimental: ”Archivo FastQ (147MB)”

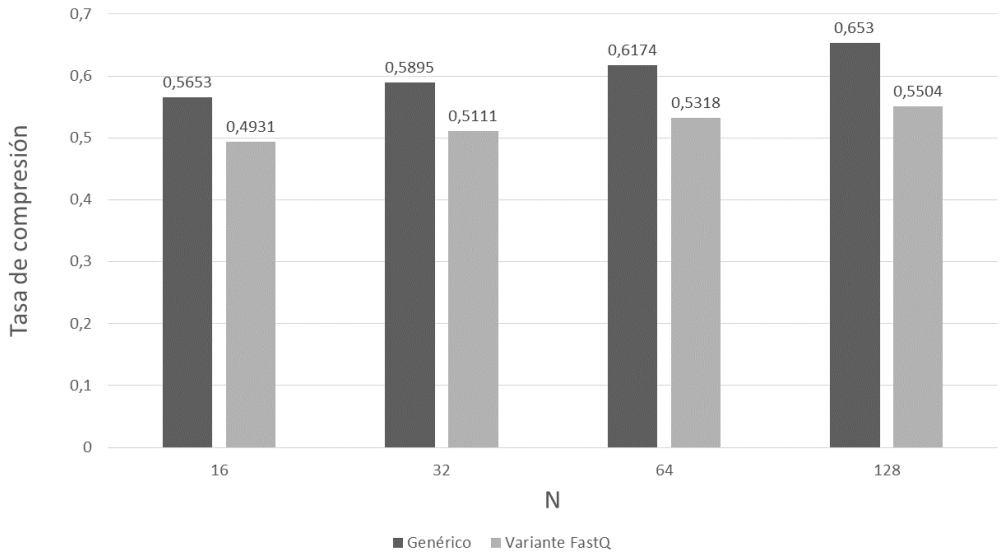

Fig. 17. Comparación de métodos genérico y adaptado con tasa de compresión en función de N (Archivo 147MB

Aquí se da la misma situación que en el archivo previamente analizado, una mayor compresión en general donde la mayor tasa de la nueva variante no supera la menor de la original. Además utilizando Zip se logró una tasa de 0.384.

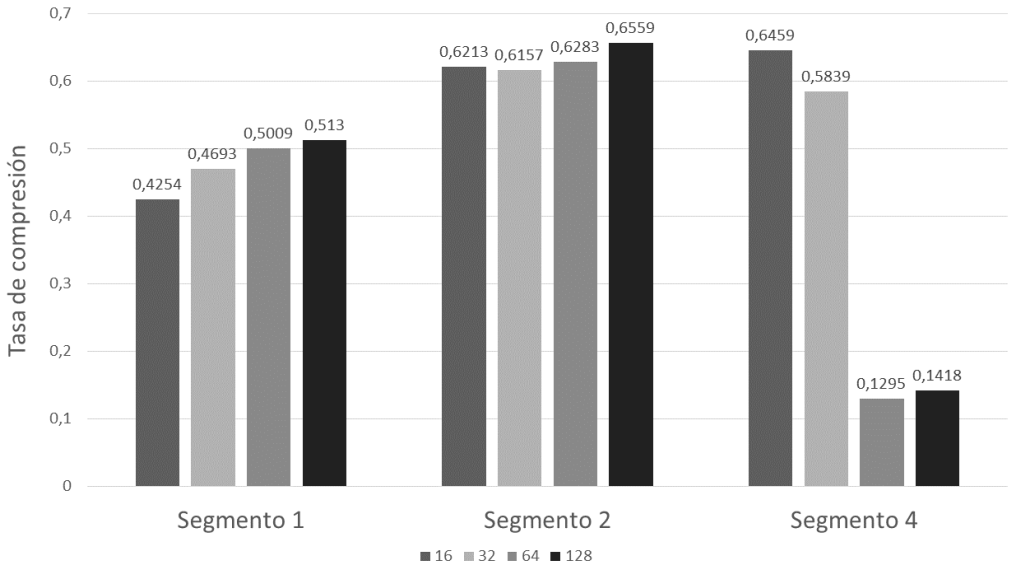

Fig. 18. Compresión según N para cada segmento (Archivo 147MB)

Aparecen resultados similares con respecto también a la compresión por segmento. Se ve cómo el segmento con mayor tasa de compresión en el archivo previo también lo es en este, lo mismo pasa con los demás segmentos, a excepción de los primeros valores obtenidos para el segmento 4 (esto último puede haberse dado al haber un gran número de subsecuencias de tamaño mayor al *N* utilizado del cual el programa se esta perdiendo de encontrar).

5 Conclusiones

Durante la fase de análisis se observaron distintos resultados al variar el tamaño de la ventana, el tamaño del archivo y la distribución de sus caracteres.

Al aumentar el tamaño del archivo junto con el de la ventana se obtuvo en general una menor tasa de compresión. Esto puede argumentarse gracias al teorema visto en [2], donde se explica cómo a medida que N al igual que la secuencia a comprimir aumenta, la cantidad de bits por símbolo se acerca a su tasa de entropía, lo cual llevaría a codificar unívocamente dicha secuencia con (cerca de) la menor cantidad de símbolos posibles (debido a la cota inferior de Shanon ([3])). En pocas palabras ,es normal pensar que mientras mayor es nuestra ventana y archivo más eficiente será nuestra compresión.

Se pudo ver, independientemente del teorema anteriormente mencionado, cómo una ventana más grande no siempre significa una menor tasa de compresión. Se observó que al tomar un tamaño potencia de 2, se obtendría una tasa considerablemente menor a un tamaño un poco mayor que no lo fuera. Además, aun siendo potencia de 2, no era extraño encontrarse con un tamaño que siendo menor a otro obtenía mejores resultados, dándose una impresión inicial de que en este se encontraría el mínimo global, finalmente verificándose que era este local al encontrar un menor valor para tamaños distántemente mayores.

Analizando la tasa de compresión a medida que se avanzaba en el archivo se vio un comportamiento similar en cada caso, el cual fue de crecimiento monótono que parecía lentamente converger. Suponiendo que en los archivos analizados había la misma posibilidad de hallar una coincidencia en cada momento es normal suponer que se daría esta situación, donde la tasa inicial se vea condicionada por la codificación de las primeras y más pequeñas etapas y/o coincidencias, y luego siga regularizándose hasta adecuarse a la tasa de compresión promedio de cada subsecuencia recorrida por el algoritmo.

Fue comparada la compresión de archivos Zip, donde se vio que la diferencia entre su tasa y la lograda con este programa era comparablemente grande. Además, curiosamente se descubrió que al comprimir una secuencia con el programa para luego hacerlo con Zip se conseguiría una tasa mayor a la lograda solo comprimiéndola con Zip. El por qué de esto es que el programa transforma una codificación de un alfabeto normalmente acotado en uno con cualquiera de los 256 caracteres ASCII, donde estos caracteres no tienen por qué seguir los patrones

que secuencias como las que se dan en obras literarias.

Con respecto a los valores de N que aparanetaban optimizar la tasa de compresión, al observar el esquema realizado en 1), donde se analiza la cantidad de bits utilizados en función a cada parámetro, vemos que el valor de N solo interfiere a la hora de representarse los largos y offsets, además de los primeros N dígitos. Dado que la cantidad de dígitos estudiados fue incomparablemente mayor a N, se descartará la posibilidad de que estos sean la causa principal. A partir de aquí la única posibilidad es que el problema surja de la representación de los largos de las coincidencias. Se sabe que al elegir un N más grande el algoritmo buscará una mayor cantidad de coincidencias, el inconveniente con la implementación del programa es que permitir esto implicará aumentar la cantidad de bits requeridos para expresar cada largo, por lo que si se aumenta el valor de N pero no se encuentra un número de coincidencias que respalde este sacrificio la compresión será menos efectiva.

Posteriormente al estudiar un archivo en donde la distribución dada minimizaba la capacidad de compresión del programa, se vio que si no se eligen cuidadosamente los parámetros y archivos manipulados pueden obtenerse resultados dañinos para la memoria del sistema donde se utiliza. Esto expone uno de los problemas a los que uno debe enfrentarse si quiere lograr un producto final bien logrado.

Finalmente, el estudio de archivos FastQ mostró un gran cambio con respecto a la tasa de compresión obtenida mediante la versión original del programa. Más allá de eso, su tasa de compresión seguía siendo considerablemente grande en comparación a formatos de compresión actuales como lo es Zip. Es imporante también resaltar que una mayor tasa de compresión en general también supone un aumento en el tiempo que tarda el programa en comprimir un archivo (obviando el tiempo que se tarda en transferir los segmentos manipulados), ya que al encontrar coincidencias de mayor tamaño el programa necesita buscar menos de ellas.

Se vio además que cada segmento tenía una tasa notablemente distinta. El segmento 4 tiene en general una menor tasa de compresión, esto se debe a que usualmente en este se tienden a escribir comentarios casi idénticos, aunque no necesariamente tiene por qué darse esta situación. Luego, el segmento 1 se posiciona entre el 2 y el 4, esto probablemente se deba a que su alfabeto es de menor tamaño, lo cual significa que el programa será capaz de representar sus caracteres con menos bits y de encontrar más coincidencias para un mismo N.

Con respecto a la tasa a medida que avanza la compresión (16), aunque sea un resultado evidente, también permite replantearse la afirmación anteriormente concluida sobre el comportamiento esperado de este tipo de gráficas. Es verdad que la compresión del archivo tienede a ser monótona y luego a estabilizarse, sin embargo, esto solo se da debido a que los caracteres de cada archivo experimentado siguen una distribución similar (por ejemplo, los libros escritos en inglés, lengua que aunque amplia recurre muchas veces a patrones similares y la repetición de palabras). Si se tratasen secuencias radicálmente distintas unidas entre si, sería normal esperar que un cambio en la tasa de compresión al momento de alcanzar cada distinta secuencia.

Como opinión, la implementación de este tipo de algoritmos no requiere de un gran conocimiento de programación, aunque si de tiempo para implementarlo. El programa no parece estar al nivel de métodos conocidos de compresión y hay situaciones como las compresiones de mayor tamaño que la del archivo original y probablemente otras aun no encontradas que me llevarían a no utilizar el programa con fines más allá que académicos. Más allá de esto, creo que su desarrollo fue una experiencia positiva, que sin duda ayudó a ver de otra forma cosas que alguien nacido en la era digital no suele cuestionarse, además de adentrarme un poco más al mundo de la programación y manejo de datos.

References

[1] Tarea_1. (2021, April). Álvaro Martin. https://eva.fing.edu.uy/pluginfile.php/338330/mod_resource/content/1/Tarea_1.pdf

[2] charla5LZ-LZ77. (2021, April). Gadiel Seroussi. https://eva.fing.edu.uy/pluginfile.php/29106/mod_resource/content/3/Diapositivas/2021/charla5LZ-LZ77.pdf. Society for Industrial and Applied Mathematics.

[3] Fuentes con distribucion conocida. (2021, March). Álvaro Martin. https://eva.fing.edu.uy/pluginfile.php/333145/mod_resource/content/2/FuentesConDistribucionConocida.pd