Digital Communication 2018-19
Lempel-Ziv Compression

Report for the Lempel-Ziv 77 Compression Algorithm

Introduction

Lempel-Ziv 77 (LZ77) is a lossless dictionary-based algorithm for sequential data compression. The dictionary is kept by a sliding window during compression. The sliding window is a portion of the previously encoded sequence, which consists of two parts:

- A search buffer that contains a portion of the recently encoded sequence, and
- A lookahead buffer that contains the next portion of the sequence to be encoded.

In this report, I am going to analyze the LZ77 compression algorithm from four perspective: running time of the encoder, running time of the decoder, compression ratio, and comparison with LZ78 algorithm.

Three test files are used for testing purpose. They are alice29.txt (~149KB), asyoulik.txt (~123KB), and lcet10.txt(~417KB).

Running time of the encoder

```
while compression not complete:
    get a reference (distance, length) to longest match
    if length > 0:
        return (distance, length, next Character)
        shift the window length+1 positions along
    else:
        return (0, 0, next Character)
        shift the window 1 position along
    end if
end while
if match:
    add 1-bit flag, followed by $\log_2(W+1)$ bits for distance, $\log_2(L+1)$ bits for length, and 8-bit character
else:
    add 1-bit flag, followed by 8-bit character
```

W: search buffer size; L: lookahead buffer size

Figure 1: The pseudocode of LZ77 encoder

To analyze the running time of LZ77 encoder, we need to calculate three quantities: maximum running time in the worst case, minimum running time in the best case, and average running time in the random case where each alphabet of original file is chosen uniformly and independently.

To answer these questions, the key implementation is to compute the distance and longest match length each time the algorithm requests those values for a position i. For each request (**Figure 1**), it traverses the search buffer that matches a prefix of $L[i..m]$. The traversal ends when no further matches are possible. The worst case is that there is no match found for all requests. As a result, the encoder has to compress the file character by character, which takes maximum running time of $c \cdot n$, where c is the unit time the algorithm taken to find a match and n is the file size in bytes. The best case is that the file is encoded by the least number of iterations. It is very hard to conclude the best case as it is affected by many uncertain factors. However, in all cases, due to the fact that the sliding window is fixed, the time to find the distance and longest match length is constant. So the entire compression algorithm runs in $O(n)$ time.

As has been started before, there are many external factors that influence the running time of LZ77 encoder. I initiated a controlled experiment to explore the impact of different window sizes, lookahead buffer sizes, and input types. In order to

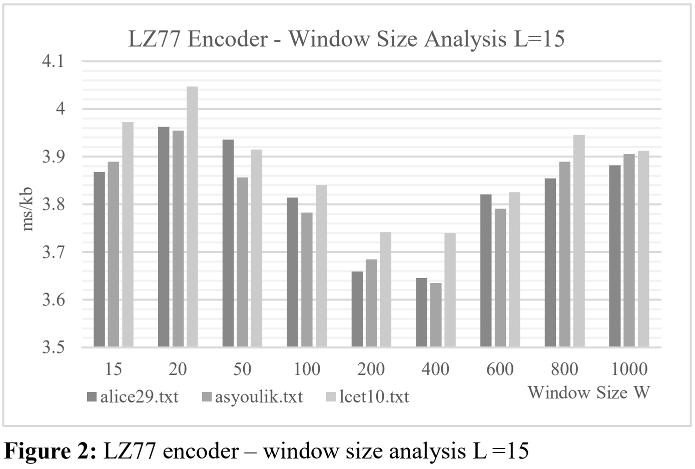

Figure 2: LZ77 encoder – window size analysis L =15

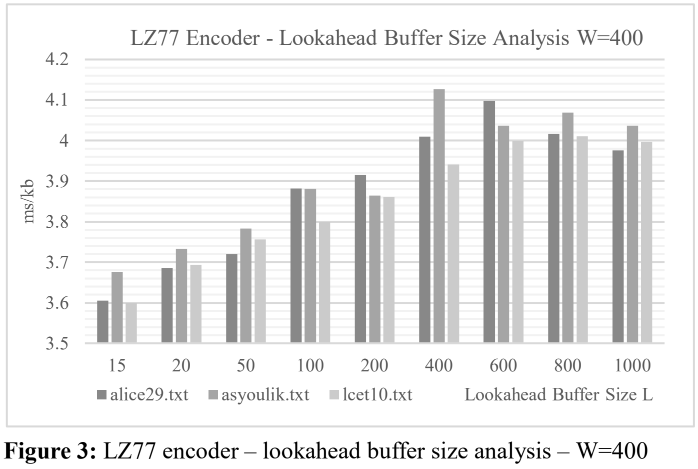

Figure 3: LZ77 encoder – lookahead buffer size analysis – W=400

compare these results in parallel among test files, the amount of time taken to compress one single kilobyte is calculated. This derived compression rate (CR) is proportional to the running time of the encoder.

Figure 2 illustrates how different window sizes affects the running time of LZ77 encoder. CR starts high and reduces gradually, reaching a local minimum at window size of 400, and keeps increasing since then. CR decreases at the beginning because as the window size increases, better matching patterns can be found in the search buffer. However, the rebound of CR at the end is caused by the settings of test environment, i.e., I explicitly set the lookahead buffer size to 15. As the window size of 400 is sufficient to find a match in most cases, larger window size will result in a waste of running time. Figure 2 well explains this situation.

Figure 3 illustrates how different lookahead buffer size affects the running time of encoder. As the lookahead buffer size increases, the encoder has to do more character comparison operations to find the longest match length, leading to a larger CR. **Figure 3** also implies that when the lookahead buffer exceeds 400, CR does not change at all. As the window size is pre-set to 400, any lookahead buffer greater than that will be meaningless since there is no way to find such a match.

Three test files of different sizes are tested using different window sizes and lookahead buffer sizes. Both **Figure 2** and **3** suggest there is no general trend between input file size and running time of the encoder.

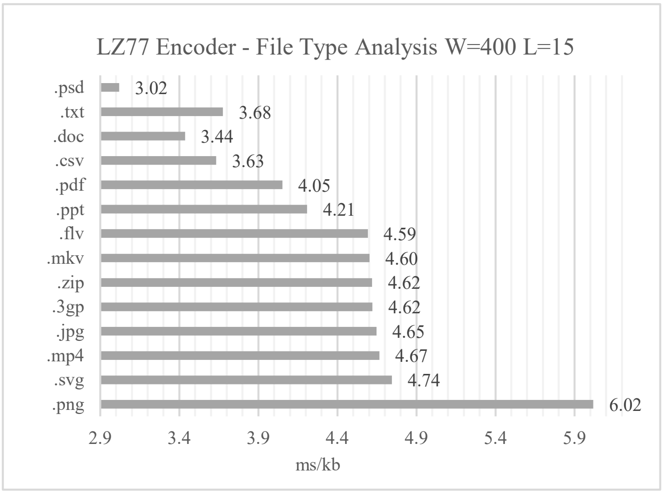

Figure 4: LZ77 encoder – file type analysis W=400 L=15

Figure 4 illustrates how different file types influence the running time of encoder. A surprising conclusion can be drawn here: the more compressed the original file is, the higher the CR is. That is, as LZ77 relies heavily on the repeated sequences, it does not have a good performance when it comes to highly-compressed file. The comma-separated values files (.csv) and text documents (.txt) do not compress any character during the storage process. A .psd file is a layered image file used in Adobe Photoshop while a .doc file is a word-based processing file created by Microsoft Word. Both of them is slightly compressed for fast I/O purpose. This explains why LZ77 encoder runs faster on these file types.

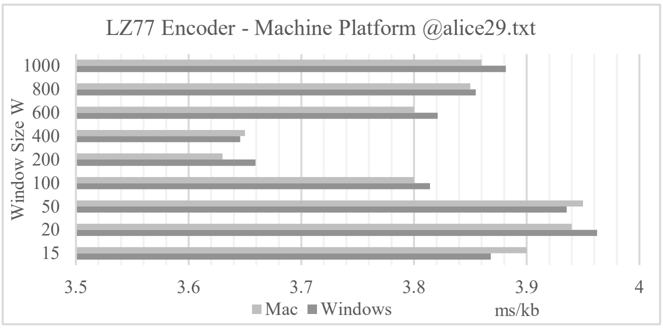

Figure 5 illustrates how different machines or operation systems affect the running time of encoder. There is no direct trend between two that in some cases, the CR on Mac platform is higher, and sometimes, it is lower than that on Windows platform. Such small difference can be explained by the computer hardware difference and other external factors.

Figure 5: LZ77 encoder – machine platform analysis @alice29.txt

Running time of the decoder

```
while decompression not complete:
    check 1-bit flag to see whether there is a
match
    if match:
        append the matched substring to output[]
        append the character to output[]
    else:
        append the character to output[]
    end if
end while
```

Figure 6: The pseudocode of LZ77 decoder

Based on **Figure 6**, we are able to tell the running time of LZ77 decompressor is solely related to the size of compressed file, which means the entire decoding process runs in $O(n)$ time, where n is the file size in bytes. However, the actual running time does vary when it comes to different use cases. That is, if the matching flag is true, the algorithm will read $9 + \log_2(W+1) + \log_2(L+1)$ bits per iteration. Otherwise, it will only consume 9 bits. On the other hand, let c be the unit time the algorithm takes to do flag check and list append operation, the maximum running time of decoder should take $(n/9) \cdot c$ time, where n is the file size in bytes. The minimum running time occurs when the decoder reads the very first character and finds a match for every iteration since then, which will take $(1 + (n - 9)/(9 + \log_2(W + 1) + \log_2(L + 1))) \cdot c$ time.

Recall that during LZ77 encoding process, the distance and best match length are stored by varied number of bits according to given window and lookahead buffer size. As a result, they play an important role to the running time of decoder. In general, given two compressed files of the same size, the one encoded using larger window/lookahead buffer size will be much faster than the other. With reference to **Figure 7** and **8**, it is easy to tell when the lookahead buffer size and window size increase, CR decreases gradually, implying faster running time of decoder. Though there is evidence of data inconsistency at the window size of 15 and 20 in Figure 7, it is caused by smaller input file size, i.e., the test files compressed using the parameters $\{W = 15, L = 15\}$ and $\{W = 20, L = 15\}$ is significantly smaller than others.

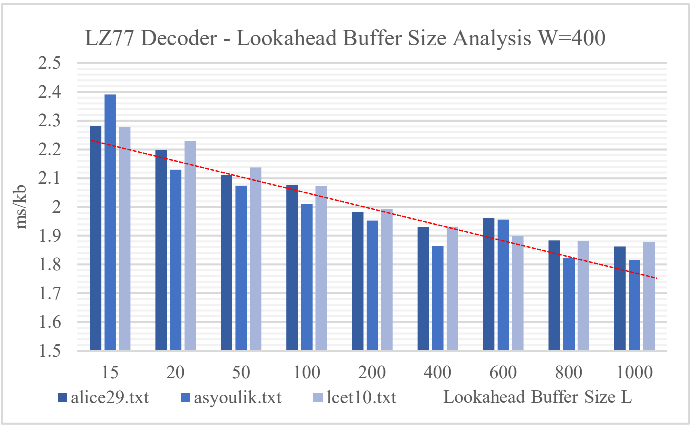

Figure 7: LZ77 decoder – lookahead buffer size analysis W=400

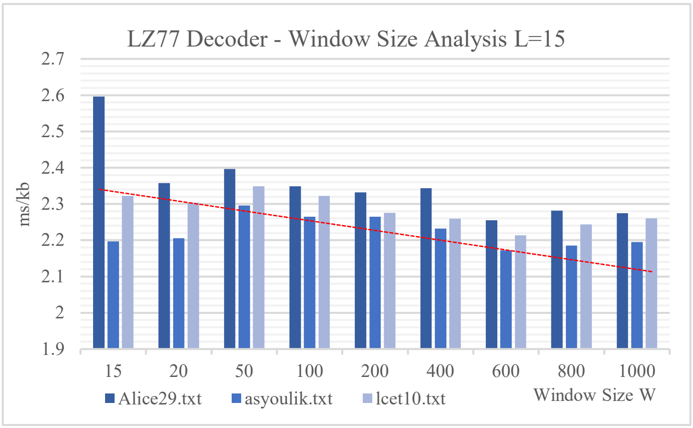

Figure 8: LZ77 decoder – window size analysis L=15

Three text files of different sizes are tested using different window sizes and lookahead buffer sizes. Both **Figure 7** and **8** suggests there is no evidence of correlation between input file size and CR.

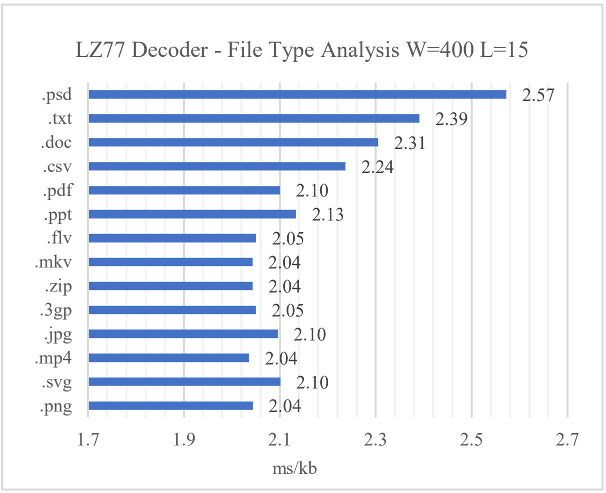

Figure 9: LZ77 decoder – File Type Analysis W=400 L=15

Besides the influence of window and lookahead buffer size, file types are a crucial determinant of running time of decoder as well. **Figure 9** lists the CR of 14 file types that cover majority of our daily needs. Surprisingly, compared to the file type analysis of LZ77 encoder (**Figure 4**), a very distinct conclusion can be drawn: the more compressed the original file is, the lower the CR is. As has been started before that LZ77 does not favor compressed file types as it has to encode them character by character, resulting in low compression ratio, this limitation becomes a big advantage during decoding process as the decoder only needs to append the character to the output buffer. Thus, compressed file types will have a much better running time than others.

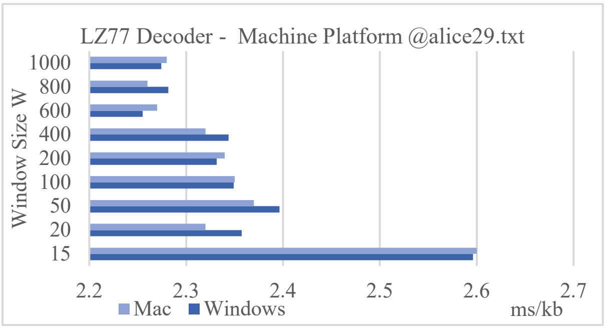

Figure 10 illustrates how different machines or operation platforms affect the running time of decoder. Similar to the encoder, there is no general trend between the running time of decoder on Windows platform or Mac platform, but they follow the same fitness curve. The small time difference can be well explained by external factors like computer hardware difference or test environment.

Figure 10: LZ77 decoder – machine platform analysis @alice29.txt

Compression ratio

Till now, I have analyzed the running time of LZ77 encoder and decoder and found how different window sizes, lookahead buffer sizes, and file types would influence them. However, these factors also co-relate to the compression ratio, where compression ratio is equal to the original file size divided by compressed file size.

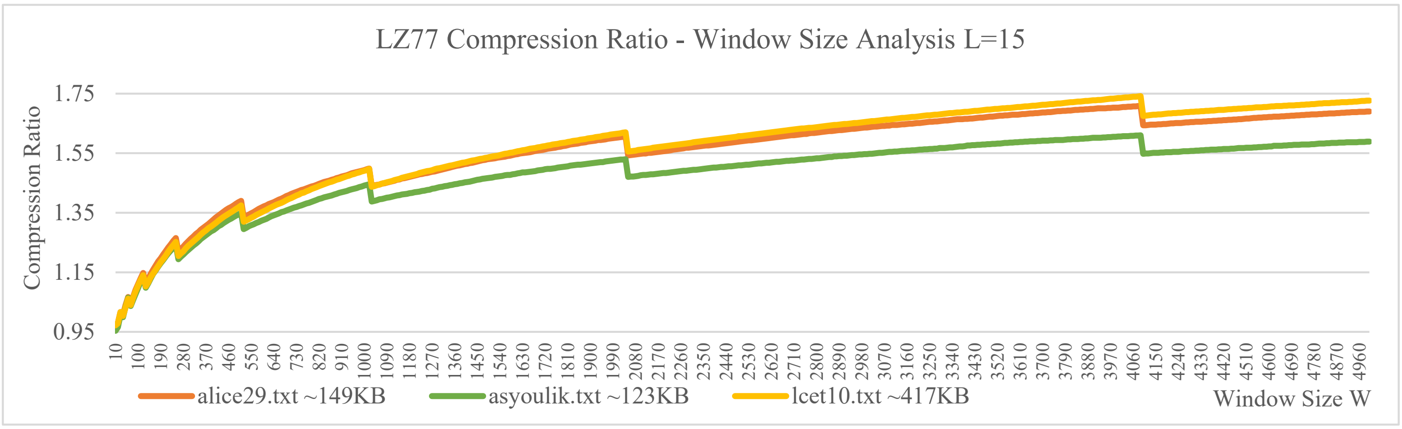

Figure 11: LZ77 compression ratio – window size analysis L=15

Figure 11 shows the compression ratio of three text files compressed using a wide range of window size from 10 to 5000. As you can see, the compression ratio gradually increases, which means the compressed file becomes smaller and smaller. However, we can also notice that there exists some dramatic drops of compression ratio at the window size of all powers of 2. This is because whenever the window size reaches another power of 2, the algorithm has to assign a new bit to fit the change. By continuing increasing the window size, it will benefit the longest match length searching process so to decrease the compression size. Hence, the compression ratio gradually increases again after powers of 2.

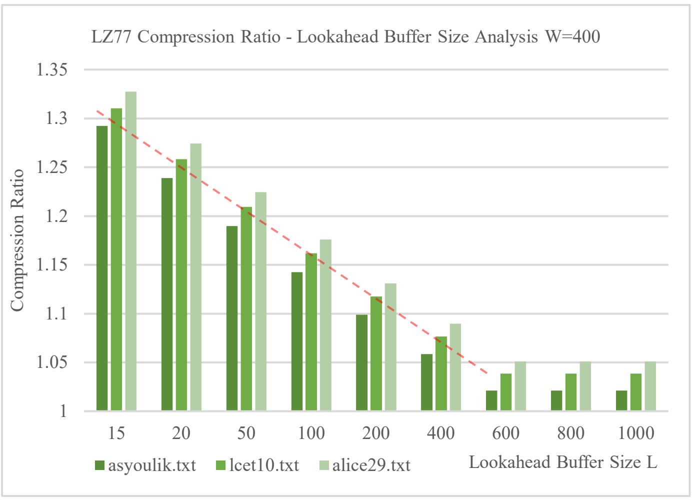

Figure 12: LZ77 compression ratio – lookahead buffer size analysis W=400

Figure 12 illustrates the compression ratio of three text files compressed using different lookahead buffer size from 15 to 1000. We can clearly tell the trend that as the lookahead buffer increases, the compression ratio decreases. Though larger lookahead buffer enables LZ77 encoder to find a match for longer unencoded sequence, it has to pay for more and more bits that represents the best match length. On the other hand, the longer the unencoded sequences is, the less likelihood the encoder finds a match. **Figure 12** well explains this phenomenon. As has been explained before, for lookahead buffer greater than 400 (window size), the compression ratio remains unchanged because there is no way to find a match if lookahead buffer size exceeds window size.

Three text files of different sizes are tested using different window sizes and lookahead buffer sizes. **Figure 11** suggests that in the long run, when the window size and lookahead buffer size become sufficiently large, larger files will result in higher compression ratios, i.e., the compression on larger files will be more effective and visible.

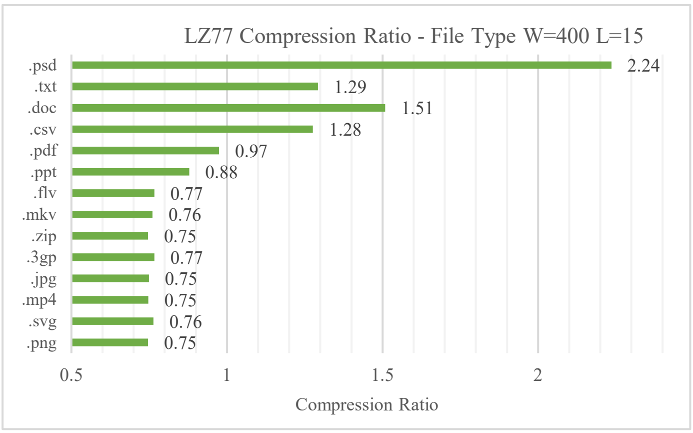

Figure 13: LZ77 compression ratio – file type analysis W=400 L=15

Figure 13 lists the compression ratio of different file types, which also reflects what we've analyzed in the previous part: the more compressed the original file is, the lower the compression ratio. As majority of video and image file format is already compressed for efficient storage purpose, their compression ratios are always below 1. This is not a good performance for a compression algorithm as the compressed file is actually greater than the original one. However, for non or slightly compressed file format including .psd, .txt, .doc, and .csv, LZ77 compression algorithm gives a relatively positive result.

Comparison with LZ78 algorithm

Similar to LZ77, LZ78 is another lossless dictionary-based data compression algorithm. However, LZ78 algorithms achieve compression by replacing repeated occurrences of data according to a dictionary maintained by the input data stream.

LZ77	LZ78
The LZ77 algorithm focuses on past data, i.e., LZ77 tries to backward search for unencoded sequence of lookahead buffer iteratively to find the best match length pattern.	The LZ88 algorithm focuses on future data, i.e., LZ78 tries to forward scan the lookahead buffer and match it against the dictionary it maintained.
The LZ77 algorithm is slower than LZ78, i.e., LZ77 can output good results when the lookahead buffer and window size is sufficiently large, but large parameters will also dramatically increase the running time of encoder/decoder.	The LZ78 algorithm is faster than LZ77, i.e., LZ78 maintains a dictionary by the input data stream, the dictionary lookup operation is much faster than byte string comparison.
The output buffer of LZ77 is a tuple [d, l, m] where d is the distance to current position, l is the best match length, and m is the next character to be encoded.	The output buffer of LZ78 is a tuple [i, m] where i is the index where the dictionary stores the information and m is the next character to be encoded.

In order to investigate their performance difference, the researchers Amit Jain and Kamaljit I. Lakhtaria did a comparative study of dictionary-based compression algorithms on text data. The following table is their experimental results that compares the performance of LZ77 and LZ78 algorithms in terms of bits per character (BPC) and compression ratio (CR).

File Name	File Size	LZ77		LZ78	
		BPC	CR	BPC	CR
Bib	111261	3.75	2.13	3.95	2.03
Book1	768771	4.57	1.75	3.92	2.04
Book2	610856	3.93	2.04	3.81	2.10
News	377109	4.37	1.83	4.33	1.85
Obj1	21504	5.41	1.48	5.58	1.43
Obj2	246814	3.81	2.10	4.68	1.71
Paper1	53161	3.94	2.03	4.50	1.78
Paper2	82199	4.10	1.95	4.24	1.89
Progc	39611	3.84	2.08	4.60	1.74
Prog1	71646	2.90	2.76	3.77	2.12
Progp	49379	2.93	2.73	3.84	2.08
Trans	93695	2.98	2.68	3.92	2.04
Average		3.88	2.13	4.26	1.90

Table 1: Comparison between LZ77 and LZ78 – BPC: bits per character; CR: compression ratio

According to Table 1, as we can see, LZ77 uses less bits to store one character than LZ78, thus having a higher compression ratio on average. Overall, LZ77 outperforms LZ78 by 12.1%. However, every coin has two sides. In their experiments, to make LZ77 algorithm output the best compression, the window is set as big as possible (but not bigger than the text). As has been discussed before, the window size plays an important role to the running time of LZ77 encoder, especially for large size files. Hence, though LZ77 outputs better result, its running time limits its application in real life.

Bibliography

Ferreira, A., Oliveira, A. and Figueiredo, M. (2009). *Time and Memory Efficient Lempel-Ziv Compression Using Suffix Arrays*. [online] Arxiv.org. Available at: https://arxiv.org/pdf/0912.5449.pdf [Accessed 14 Feb. 2019].

Jain, A. and I. Lakhtaria, K. (2013). *Comparative Study of Dictionary-based Compression Algorithms on Text Data*. [online] Pdfs.semanticscholar.org. Available at: https://pdfs.semanticscholar.org/f803/eb4ec22c42576cc00b5d0b8ada03ae4ab80a.pdf [Accessed 14 Feb. 2019].

Shor, P. (2005). *Lempel-Ziv*. [online] Www-math.mit.edu. Available at: http://www-math.mit.edu/~shor/PAM/lempel_ziv_notes.pdf [Accessed 14 Feb. 2019].

Zeeh, C. (2003). *The Lempel Ziv Algorithm*. [online] Fenix.tecnico.ulisboa.pt. Available at: https://fenix.tecnico.ulisboa.pt/downloadFile/3779571247713/LempelZiv_Zeeh.pdf [Accessed 14 Feb. 2019].