# Perhaps Size Does Matter

Lempel-Ziv 78 Compression

Arnav Nepal

March 13, 2023

## 1 Introduction

In this assignment we were tasked with implementing our own version of the Lempel-Ziv 1978 (LZ78) compression method. LZ78, named after a paper in 1978 by creators Abraham Lempel and Jacob Ziv, used a dictionary to compress data in pairs. It utilized the existence of patterns in language to compress words whenever there is a repeated pattern of letters in a text. In our implementation, we used a prefix tree, or "trie", to store existing word for easy retrieval at a later time. On the other hand, we used a simple look up table of a pair of values to efficiently look up symbols for decompression.

## 2 Analysis and Data

| file | original | compressed | ratio |
|---|---|---|---|
| navy.txt | 1517 | 1136 | 25.12% |
| bee.txt | 55316 | 31217 | 43.57% |
| meta.txt | 119743 | 61653 | 48.51% |
| small.txt | 1010000 | 956543 | 5.29% |
| kjv.txt | 4457889 | 206918 | 53.57% |
| medium.txt | 10100000 | 9557813 | 5.37% |
| large.txt | 101000000 | 95574999 | 5.37% |

The above table displays a few statistics of LZ78 compression. I used 6 different files of different types to test how well LZ78 works under different conditions. The details of the files are explained below:

- navy.txt : the well known "navy seals copypasta", a short, 1.5 KB text
- bee.txt : the script of the "Bee Movie"
- meta.txt : All of Franz Kafka's *Metamorphosis* obtained from Project Gutenberg

– kjv.txt : The entire King James Bible obtained from Project Gutenberg

– small.txt : 1 MB file of random characters

– medium.txt : 10 MB file of random characters

– large.txt : 100 MB file of random characters

When putting together the details of each file and the statistics in the table, an intriguing pattern emerges: namely, LZ78 excels in natural language, as exemplified by it's relatively good performance in the navy, bee, and meta text files. On the other hand, it performs poorly in the randomly generated text files, no matter the size.

## 3 Lessons Learned

This assignment worked to bring together everything learned over the quarter. We were given function explanations and minimal psuedocode for encoding and decoding, leaving everything else up for us to personally implement. This was also out first time working with the low level input/output functions such as open() and close(). I learned a lot in this assignment in the process of implementing each abstract data type and module.

### 3.1 Compression: LZ78

The big thing I learned in this assignment was about compression. Specifically, we learned about LZ78 compression, an older but relatively simpler compression algorithm. LZ78 works through pairing "words" in pairs with a "code" within a dictionary. The word containing letters functions as the key to the dictionary while the code is the associated value. Each time the compression algorithm comes across a letter, it checks if the letter exists within the dictionary. If it does not, it is added to the dictionary at the next available code. It is then written to the output as $< 0, c >$, where 0 signifies no previous record of that letter and c a character/word. When repeated letters or sequence of letters come up, they are added to the dictionary with the next unique letter and the associated code of the repeated letter. For example, let there exists a word 'a' at code 5 in a dictionary. When LZ78 comes upon another 'a' in text, the algorithm looks at the next letter, which might be an a new, unique char 'x'. LZ78 will write the code of the repeated letter along with the new character to output, like $< 5, x >$. When decompressing, the algorithm then knows to look at index 5 for the word associated with x. Once found, it combines the 2, and writes out 'ax' to the output.

### 3.2 Low Level I/O

In this assignment, we were exposed to low level system call functions for input/output. These functions, while less convenient then wrapper functions like printf and scanf, offer a greater degree of control to the user. We used these functions to perform all of our input and output for this program, as we needed to obtain or write out a specific number of bytes for each process. In the process of using these new functions, I learned how complex system calls can be. For example, we had to loop calls to read() and write() in our I/O module because the operating system often interrupted these calls to handle other processes.

## 4 Conclusion

LZ78 seems to be a good data compression algorithm *under certain conditions*. When compressing natural language texts with relatively low entropy, LZ78 can reach high levels of compression, up to 50% (as per my tests). On the other hand, when confronting files with high levels of entropy such as those containing randomly generated characters, LZ78 seems to face problems is getting high amounts of compression. This is not too surprising; LZ78, like many other compression methods, takes advantage of patterns in text for compression. Therefore, it is bound to face problems when compressing texts with a lack of patterns.