Assignment 6
WRITEUP
Prof Veenstra
Nguyen Vu

In this assignment, I implement the Lempel-Ziv Compression (LZ78) Algorithm for encryption and decryption data in C programming language and in this WRITEUP.pdf, I will talk about everything I learned while doing this assignment.

1. General knowledge gain

1. Tries: I gain a deep understanding of how tries work as an efficient information retrieval data structure. A trie is essentially a prefix tree where each node represents a symbol or character in the input data and has n child nodes, where n is the size of the alphabet being used (typically 256 for ASCII characters). As part of implementing the LZ compression algorithm, I learn how to efficiently traverse tries, manipulate and store words in the trie, and manage the overall memory usage of the data structure.

2. Word Tables: I learn how to create and manage Word Tables, which are lookup tables stored as structs used for quick code to word translation during decompression. Because there can be no more than 216 - 1 codes (one of which is reserved as a stop code), the size of the Word Table is limited to UINT16_MAX, the maximum value of a 16-bit unsigned integer. As part of implementing the LZ compression algorithm, I learn how to create and manage these lookup tables to ensure that decompression is efficient and effective.

3. I/O: Efficient input/output (I/O) operations are crucial for the success of the LZ compression algorithm. To optimize I/O performance, I learn how to implement buffering, which involves storing data in a buffer (an array of bytes) and processing it in blocks of 4KB at a time. I gain deep understanding of memory management, as well as expertise in optimizing read and write operations to minimize bottlenecks and ensure maximum throughput.

2. Functionality of LZ78 compression

The LZ78 algorithm is a data compression technique that operates by identifying and exploiting recurring patterns within the input data. Its primary goal is to represent the input data in a more concise and efficient manner, using a sequence of codes that correspond to entries within a dynamically-created dictionary.

The LZ78 algorithm involves three primary steps. First, an empty dictionary is initialized. Second, the input data is scanned from left to right, with the algorithm seeking to identify the longest sequence of characters that corresponds to an existing sequence in the dictionary. If a match is found, the algorithm outputs a code that references the dictionary entry for that sequence, and continues scanning the input data from the end of the matched sequence. If no match is found, the algorithm outputs a new code that represents the current input character, and adds the new sequence (including the current character) to the dictionary. Finally, the output of the algorithm is a sequence of codes that represent the input data using entries in the dictionary.

The dictionary used by LZ78 in this implementation is a trie, which provide efficient lookup and insertion of entries. This allows the algorithm to work quickly and effectively on a wide range of input data types, including streaming data that is processed as it is received.

3. How the efficiency of your compression changes with entropy. Explain the relation.

When we compress data using the LZ78 algorithm, we look for repeated patterns of characters in the data and replace them with shorter codes. The more often we find these repeated patterns, the more effective the compression will be.

However, the amount of repetition in the data is affected by something called entropy. Entropy measures how much randomness or unpredictability there is in the data. If the data is very random, there won't be many repeated patterns to find, and the compression won't work as well. On the other hand, if the data has a lot of predictable patterns, the compression will be more effective.

So, the relationship between LZ78 compression and entropy is that high entropy data is harder to compress because it has less repetition, while low entropy data is easier to compress because it has more repetition. It's important to keep this in mind when using the LZ78 algorithm, because it will perform better on some types of data than on others.