WRITEUP

Ananya Batra

March 13, 2023

In assignment 6, I learned about data compression, which means reducing the amount of bits needed to represent the data. This is important because compressed data can be transferred faster and stored easier. In this assignment specifically, I implemented LZ78 compression. A user can compress a file with encode.c, which implements LZ78 compression. LZ78 compression is a lossless compression (compression algorithms that do not lose data, meaning that decompressed information exactly matches the original information) algorithm that compresses a file by storing a sequence of bytes (words) into a dictionary that gets built during the compression process. Words are denoted by codes in the dictionary. The compression algorithm works by replacing any future occurrence of a word that exists in a dictionary with its code. These words are stored in a trie.

In this assignment, I got to learn about trees in general and tries in specific. A tree is a data structure that organizes information in a tree-like format. It is typically composed of nodes, which contains some value. The root node is the starting point of a tree and can have child nodes (nodes connected to a parent node and only accessible through their parent). Each child node can also have its own child nodes. A trie, also called a prefix tree, is a specific type of tree usually used to store strings. Each TrieNode contains a symbol and n child nodes (256 in our case). The symbol in our case is the code. The children represents the possible next characters to form the prefix. If a child is null, that means the character it's associated with is not a part of that subtree. Tries are incredibly useful because it allows for grouping words based on common prefix.

Now that the file has successfully been compressed, a user can decompress a file (provided that it was compressed with encode.c) with decode.c. Decompression works by translating codes to words (using a lookup table called a Word Table which is just an array of words).

Both the encode and decode programs perform read and writes in blocks of 4KB. io.c contains functions in order to facilitate efficient reading and writing. While implementing the functions in io.c, I learned more about about input, output, and buffers. Buffers are temporary storage for data to be written out or read in that results in more efficient reading and writing operations (because reading and writing 1 byte at a time is slow). Most of the file reading and writing functions we've used so far have handled the buffering for me, but in this assignment, I had to handle it myself. For this assignment I used read() to read in data from infile into a buffer and write() to write out data from a buffer into an outfile. I also had to be mindful of my position inside the buffer to make sure I was not overwriting preexiting information or writing the same information out multiple times.

Our compression does depend on the entropy of the data. In our case, the entropy represents the measure of uncertainty of the occurrence of an event. A file with low entropy means that it mainly consists of a few characters, repeated many times. A file with high entropy has a higher distribution of characters and fewer occurrences of each character. LZ78 Compression will in general perform better on higher entropy data since the entire idea behind the compression is to recognize recurring patterns (prefixes) in the text and compress it by substituting a code in place of the sequence (the code is likely fewer bytes than the sequence). This means that the algorithm will perform better on files with repeating characters and sequences as well as files that are longer (allowing for more opportunities for sequences to reoccur). On the other hand, it will perform worse on files with little to no recurring sequences. Additionally, it will also perform worse on shorter files simply because there are fewer opportunities for sequences to repeat, meaning that not much compression can take place with the LZ78 algorithm.