WRITEUP - ASGN6

Viveka Agrawal

CSE13S, Winter 2023

In this assignment, I learned how to implement the Lempel-Ziv (LZ78) algorithm. The algorithm is used for compressing data using variable bit-length codes and fixed 8-bit sized symbol pairs. The program reads in the data to compress and writes out the compressed data to an outfile. Next, the program reads in the compressed data file. It then decompresses the input file using the decompression algorithm and writes out the decompressed data to the outfile. The program is implemented in 2 parts: an encoder and a decoder. The most costly part of compression is checking for existing words. We used the trie structure to store the words as many of the words needed to be stored are prefixes of other words. Therefore, the trie structure is the most optimal method to store the words. We used a WordTable during decompression as decompression needs a look-up table for quick code-to-word translation.

The main idea behind the Lempel-Ziv compression algorithm is to represent repeated patterns in data using pairs which are each comprised of a code and a symbol. The code is an unsigned 16-bit integer and a symbol is an 8-bit ASCII character. We initialize the dictionary with an empty word of string length 0 at index EMPTYCODE (1). Next, we look at the first word in the input file and check if this word has already been seen before. Since this word does not exist in the dictionary, we store it at index STARTCODE (2) and store the previously seen word to be the empty word and output the pair (EMPTYCODE, 'a') to the output file. We repeat this process for the rest of the words in the infile. If the word does not exist in the dictionary, we store it at the next index in the dictionary and store the pair (code, symbol) in the outfile. Once all the words in the infile have been read-in, we output the final pair (STOPCODE, 0) to indicate the end of compression.

The Lempel-Ziv compression algorithm is most effective when the entropy in the data to be compressed is low. Since the algorithm represents repeated patterns in data using pairs, a higher number of repeated patterns in data is preferred. With a higher number of repeated patterns, the size of the compression dictionary will be smaller. As a result, the number of (code, symbol) pairs written to the outfile will be smaller and the compressed output file size will be much smaller than the uncompressed input file size. If the entropy is high, this means that there are a less number of repeated patterns in the data, so the size

of the compression dictionary will be larger. A larger number of (code, symbol) pairs need to be written to the output file. As a result, the compression ratio will be smaller.

The autograder on git shows no errors for me for scan-build. Here is the proof:

```
viveka@viveka-VirtualBox:~/cse13s/asgn6$ scan-build make
scan-build: Using '/usr/lib/llvm-15/bin/clang' for static analysis
make: Nothing to be done for 'all'.
scan-build: Analysis run complete.
scan-build: Removing directory '/tmp/scan-build-2023-03-09-175150-2409-1' because it contains no
reports.
scan-build: No bugs found.
```