Matthew Hrydil

mbh38

CS1501 – Assignment 3 Writeup

Summer 2019

The goal of this assignment was to modify the author’s code for LZW compression to improve upon its compression amounts. The way this was done was to incorporate variable length codewords, as well as the ability to reset the dictionary when it became full. Both of these improvements allow greater compression in certain circumstances.

Generally speaking, the option to reset the dictionary didn’t have an effect on the smaller file sizes, since the dictionary never filled up for those files in the first place. However, it did affect the compressed size of some of the larger files. Using dictionary reset for “all.tar” resulted in a great improvement over the variable code length version without the reset. But for the other large files, using the dictionary reset actually increased the size of the compressed file. This is likely because when it reset the dictionary, it lost all the strings that it had already begun compressing, and it had to begin re-compressing those strings again. This explains the good compression ratio for “all.tar”, since there were many different types of files within it, it was able to use different compression strings for the different files within “all.tar”.

The worst results for the LZW compression algorithms came from the .gif and .jpg files. All three versions of these two images resulted in an expansion over the original size. The Unix compression made very little improvement over the original size of the .gif file, and the .jpg was the exact same size after being compressed using the Unix command.

The bitmap images worked much better for the LZW compression algorithm. The author’s compression algorithm did not work well for the winnt256.bmp file, but it was able to compress the wacky.bmp file to about .5% of the original size! Both of the variable code-length versions resulted in the same compression ratios because the dictionary never filled for any of the .bmp files. The variable code length version was able to compress the “bmps.tar” file to 7% of their original size. This is likely because .bmp files have long strings of the same character, and the LZW compression was able to capitalize on those strings.

For the small .txt and .doc files, both LZWmod compression algorithms and the Unix compression algorithm resulted in roughly the same compression ratio. The ratio was between 30-50% for each of the .txt and .doc files. Because there is more entropy in the .txt and .doc files than there is in a .bmp file, they weren’t able to be compressed as much, but overall they still compressed a decent amount over their original size.

The .exe file performed slightly worse than the .txt files, but it still resulted in about a 60% compression ratio for the variable code length compression algorithms.

To sum things up, the variable code length algorithms out-performed the author’s 12-bit code length algorithm for every file. For many of the files, the compression ratios for the variable length codewords were comparable to the Unix compression ratios. As expected, the dictionary reset benefitted the all.tar file since it was a large file with many different file types. Lastly,

