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. (With reset the compression ratio was 2.57:1 compared to 1.69:1 without dictionary reset.) For the other large files where the dictionary reset occurred, the differences were less drastic. In some cases, using dictionary reset increased the size of the compressed file, but overall the compression ratios were comparable. Since all.tar contained many different file types, it makes sense that the compression would benefit from dictionary reset. As the dictionary became full, it could reset the dictionary and begin optimizing the strings for the different file types. For the .txt files, this doesn’t result in many improvements since the general file structure is consistent throughout.

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. These were the only files that did not show any improvement with the variable code length LZW compression.

The bitmap images worked much better with the LZW compression algorithm. The author’s compression algorithm did not work well for the winnt256.bmp file, but it worked very well with wacky.bmp. Both of the variable code-length versions resulted in the same compression ratios as one another because the dictionary never filled for any of the .bmp files. The variable code length version of bmps.tar resulted in a 13.67:1 compression ratio, compared to the fixed code length version which resulted in a 1.20:1 ratio. .bmp files are great for LZW compression because they include many strings of the same character. This means that a single codeword can represent many characters from the original file, greatly increasing compression. In the case of wacky.bmp, since the image was mostly white space, each of the compression algorithms yielded a compression ratio greater than 200:1!

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 2:1 and 3:1 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 respectable amount over their original size.

The .exe file performed slightly worse than the .txt files, but it still resulted in a ~1.5:1 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. For the other files, the option to reset the dictionary didn’t affect the overall compression ratios that much, especially the smaller files where the dictionary reset never occurred.