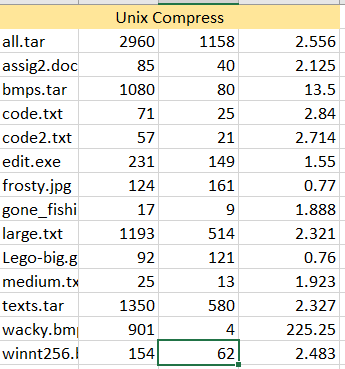
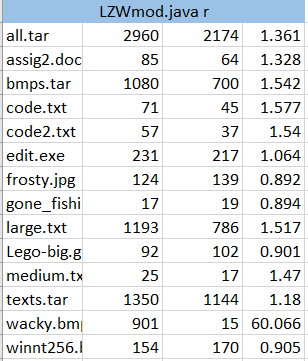
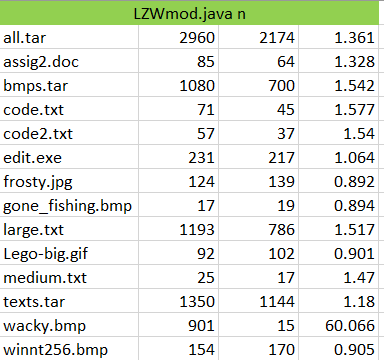
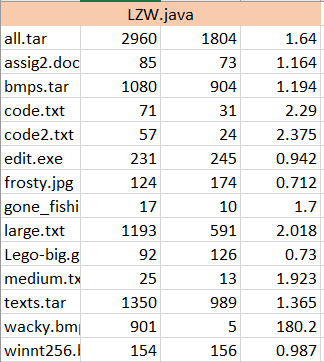
Zhengming Wang

CS1501 – Assignment 3

29 June 2019



First Column is File name

Second Column is Original Size

Third Column is Compressed Size

Fourth Column is Compression Ratio

The sizes are measured in kilobytes.

In Project 3, 4 different Compression Algorithms were compared on 14 different files to see which had the best compression ratio (original size/compressed size). The first algorithm used was the Robert Sedgewick’s own LZW variation. His version performed well, and had an average compression ratio of 14.23. Excluding wacky.bmp it had an average ratio of 1.46. LZW did best on .txt files, where it had an average ratio of 2+. The other compression algorithms also did similarly well on .txt files. This could be because .txt files have encodings that that allow for longer string patterns. Longer patterns allow for more efficient codewords which means higher compression ratios. The next algorithms were variations of my improvement on Sedgewick’s LZW. The version labeled LZWmod n wouldn’t reset the dictionary when it filled up, while LZWmod r reset the dictionary each time it filled up. The two LZWmod’s both had a better average compression than the original. Unfortunately, the modified LZW used codewords that went from 9 bits to 16 bits. Since 16 bits allow for more codewords, the dictionary reset never occurred and the two algorithms had identical values. Still one could speculate LZWmod r one compress better than LZWmod n on very large .tar files. This is because .tar files would have various files which meant different types of encodings, rendering previous codewords from the dictionary won’t work as well. This reasoning explains why most algorithms struggled to compress .gif, .jpeg, and .bmp files. These files most likely have high data entropy which results in extremely poor compression rates. It came to the point where no algorithms had a ratio better than 1:1 for these files, not even the Unix compressor, which had the highest average ratio of 2.94 (barring wacky.bmp). In some files, the original LZW was able to outperform my modified versions. This indicates that in some cases, having the standard 12-bit code word and doing nothing could be more optimal.

Wacky.bmp had an extraordinarily high compression ratio throughout all 4 algorithms. There could be a convenient encoding in the file that allows for long patterns.

I also created the extra credit compression algorithm that resets the dictionary’s contents based on a threshold. I chose the threshold for reset to be 0.9, and it was available to improve on the modified algorithms. I chose 0.9 as the threshold because I would satisfied with any compression that is 1:1 and didn’t want my compression to add more memory instead of save memory.