|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **File** | **Original File Size** | **Compressed File Size (Original)** | **Compression Ratio** | **Compressed File Size (w/o Reset)** | **Compression Ratio** | **Compressed File Size (w/ Reset)** | **Compression Ratio** | **Compressed File Size (Unix)** | **Compression Ratio** |
| all.tar | 3 MB | 1.8 MB | 1.67 | 1.8 MB | 1.67 | 1.2 MB | 2.5 | 1.2 MB | 2.5 |
| assig2.doc | 87 KB | 75 KB | 1.16 | 40 KB | 2.18 | 40 KB | 2.18 | 40 KB | 2.18 |
| bmps.tar | 1.1 M | 925 KB | 1.21 | 81 KB | 13.91 | 81 KB | 13.91 | 81 KB | 13.91 |
| code.txt | 72 KB | 31 KB | 2.32 | 25 KB | 2.88 | 25 KB | 2.88 | 25 KB | 2.88 |
| code2.txt | 58 KB | 24 KB | 2.42 | 21 KB | 2.76 | 21 KB | 2.76 | 21 KB | 2.76 |
| compress.exe | 47 KB | 38 KB | 1.24 | 32 KB | 1.47 | 32 KB | 1.47 | 32 KB | 1.47 |
| edit.exe | 236 KB | 251KB | 0.94 | 156 KB | 1.51 | 152 KB | 1.55 | 151 KB | 1.56 |
| frosty.jpg | 127 KB | 177 KB | 0.72 | 164 KB | 0.77 | 171 KB | 0.74 | N/A | N/A |
| gone\_fishing.bmp | 17 KB | 9 KB | 1.89 | 9 KB | 1.89 | 9 KB | 1.89 | 9 KB | 1.89 |
| large.txt | 1.2 MB | 605 KB | 2.03 | 502 KB | 2.45 | 572 KB | 2.15 | 523 KB | 2.35 |
| Lego-big.gif | 93 KB | 129 KB | 0.72 | 122 KB | 0.76 | 122 KB | 0.76 | N/A | N/A |
| medium.txt | 25 KB | 13 KB | 1.92 | 13 KB | 1.92 | 13 KB | 1.92 | 12 KB | 2.08 |
| texts.tar | 1.4 MB | 1 MB | 1.4 | 598 KB | 2.4 | 591 KB | 2.43 | 590 KB | 2.43 |
| wacky.bmp | 922 KB | 4 KB | 230.5 | 4 KB | 230.5 | 4 KB | 230.5 | 4 KB | 230.5 |
| winnt256.bmp | 157 KB | 159 KB | 0.99 | 63 KB | 2.49 | 63 KB | 2.49 | 63 KB | 2.49 |

As is visible in the above table, the author’s implementation is usually less effective than any of the other implementations. This is because the author’s code uses a fixed-size codeword, which quickly fills a symbol table and halts the pseudo-learning nature of LZW. Since LZW works by finding previous patterns, a variable-length codeword is more effective. The only places where this was not the case were wacky.bmp, gone\_fishing.bmp, and all.tar. Bitmap files have pixel’s color based on a number of bits per pixel. The two aforementioned bitmap files are much of the same color, and so compression is extremely effective because of the repeated patterns in the file. The file all.tar is the largest file of the bunch and is a tarball. It likely has few repeated patterns and it can be assumed that without a reset functionality, the compression algorithm wouldn’t be able to use longer patterns.

The modified implementation with the reset functionality generally performed better on larger files than the without reset implementation. This is because on smaller files, they are essentially the same program—the codebook is never filled when W=16 because the file is not large enough. In some cases, though, the reset functionality actually performed worse. This is because when we throw out the codebook, we may be getting rid of valuable patterns that are likely to appear later in the file. This is the case with large.txt, because the file is in English and words/patterns are likely repeated, the reset functionality performed worse.

The Unix implementation of LZW was very similar to the reset-enabled implementation. Based off of the “man compress” information, the Unix implementation uses variable length codewords in the same 9-16 bit way as the modified implementation. The Unix implementation also uses reset, but with one difference: it will only reset if the compression ratio is decreasing. This is useful because the codebook will be used if there are patterns useful later on in the file, which is why large.txt works better with the Unix implementation versus the reset enabled implementation.

For every implementation, wacky.bmp had the best compression ratio, with 230.5. This is because the file is a bitmap image that is mostly white and as previously mentioned, bitmap images represent pixel colors with a certain number of bits. Since the file is extremely repetitive in this way, it can be compressed extensively by any algorithm without using too many codewords.

Also, for every implementation Lego-big.gif and frosty.jpg had the worst compression ratios. The ratios were around 0.75 for every implementation except the Unix implementation, which ignores the files. The Unix implementation will ignore files that will not compress at all (meaning if they are going to expand, they will not be changed). The data in these two files, when compressed, is stored as codewords that are larger than what is originally in the file. This results in a larger output than input.