|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| FileName | Original | LZW | LZW Variable | LZW Reset(monitor) | UNIX Compress |
| Lego-big.gif | 92K | 126K | 120K | 120K | 120K |
| All.tar | 2.9M | Wouldn’t finish | 1.75M | 1.75M | 1.2M |
| Bmps.tar | 1.08M | 904K | 80K | 80K | 80K |
| Texts.tar | 1.35M | 989K | 584K | 584K | 556K |
| Assig2.doc | 85K | 73K | 40K | 40K | 40K |
| Code.txt | 71K | 31K | 24K | 24K | 24K |
| Code2.txt | 57K | 24K | 21K | 21K | 21K |
| Edit.exe | 231K | 245K | 153K | 153K | 148K |
| Frosty.jpg | 124K | 174K | 160K | 160K | 160K |
| Gone\_fishing.bmp | 17K | 10K | 9K | 9K | 8.8K |
| Large.txt | 1.2M | 591K | 491K | 490K | 503K |
| Medium.txt | 25K | 13K | 13K | 13K | 13K |
| Wacky.bmp | 901K | 5K | 4K | 4K | 3.9K |
| Winnt256.bmp | 154K | 156K | 62K | 62K | 62K |

Alex Visbisky – LZW Compression Assignment 3

**Note** – I did not get the char-by-char implementation to work.

When analyzing this table there are many trends here. The table here shows that the original LZW code (the one implemented by the authors) does not work better in any situation where compression worked. This is most likely due to the static size of the codebook; it fills up rather quickly then it just remains the same instead of adding new codewords of new length to continue compression. The variable compression I implemented seems to save comparable amounts to UNIX’s compression and the monitored compression I implemented. Monitored compression really only saves space when the codebook is filled up and/or the compression rates rise above the threshold value of 1 (comparing the rate of data read to how much it is currently being compressed). This typically will only occur when the file being encoded has used a lot of the codebook and what is already encoded isn’t efficient what is being encoded, such as large.txt that saves an extra kilobyte here. For files like Lego-big.gif LZW compression actually makes the file larger since the file format already uses lossless compression and compressing it again only increases the size. UNIX compressed the exe much better than my implementation of LZW could. I believe this is due to the difference in algorithms used. UNIX compression uses LZW as well but it may implement it better with a more closely monitored compression ratio. Frosty.jpg also increases in size, JPG is a lossy format and compressing that again is only using more space on an already compressed file. Other than those exceptions everything compressed as assumed. Text files and bitmaps seemed to have the best compression ratios, they’re raw data so it’s easy to build codebooks for that and save space since a lot of information can be repeated there. They can typically achieve saving more than half the space of the file in this assignment. For the all.tar file (I’m unsure if we were supposed to compress it or not) the author’s LZW ran on my machine for close to an hour before I decided that it either was hung up on something or it was going to continue to take way too long to achieve a result. Practically, having it take that long to compress a file of that size is unusable. UNIX compress also beat my compression for the tar file though and I believe again that this is due to the LZW implementation used in UNIX, possibly due to a better monitoring of the compression ratio.