Khalil McFarlane

Professor Khattab

Algorithm Implementation

6 November 2020

Assignment 3 Writeup

LZW Compression allows programmers to compress files using a symbol table lookup. In this Assignment, we created an LZWmod.java program and used TST.mod as our symbol table implementation. The purpose of this Assignment was to compare the compression ratios obtained from using the given LZW.java, compress.exe, and the program we made, LZWmod.java. Here is a graph that compares the original size, compressed size, and compression ratio (original size/compressed size).

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| File Name | LZW.java  (Original/Compressed/Ratio)  (KB) | LZWmod.java w/o Reset  (Original/Compressed/Ratio)  (KB) | LZWmod.java w/ Reset  (Original/Compressed/Ratio)  (KB) | Compress.exe  (Original/Compressed/Ratio)  (KB) |
| all.tar | 2,960, <1, 2,960 | 2,960, 1,751, 1.69 | 2,960, 1,151, 2.6 | 2,960, 1,152, 2.57 |
| assig2.doc | 85, 73, 1.2 | 85, 40, 2.1 | 85, 40, 2.1 | 85, 40 KB, 2.1 |
| bmps.tar | 1,080, 904, 1.19 | 1,080, 80, 13.5 | 1,080, 80, 13.5 | 1,080, 80, 13.5 |
| bogusInput.txt | 0, Read from empty stream, N/A | 0, Read from empty stream, N/A | 0, N/A, N/A | 0, No Z file made, N/A |
| bogusOutput.txt | 0, Read from empty stream, N/A | 0, Read from empty stream, N/A | 0, N/A, N/A | 0, N Z file made, N/A |
| code.txt | 71, 31, 2.3 | 71, 24, 2.96 | 71, 24, 2.96 | 71, 24 KB, 2.96 |
| code2.txt | 57, 24, 2.4 | 57, 21, 2.7 | 57, 21, 2.7 | 57, 21, 2.7 |
| edit.exe | 231, 245, 0.9 | 231, 153, 1.5 | 231, 149, 1.6 | 231, 148, 1.56 |
| Frosty.jpg | 124, 174, 0.7 | 124, 160, 0.8 | 124, 168, 0.7 | 124, No Z file made, N/A |
| gone\_fishing.bmp. z | 9, 13, 0.71 | 9, 13, 0.71 | 9, 13, 0.71 | 9, 9, 1 |
| large.txt | 1,193, 591, 2.02 | 1,193, 491, 2.4 | 1,193, 516, 2.3 | 1,193, 511, 2.3 |
| Lego-big.gif.temp | 92, 126, 0.73 | 92, 120, 0.77 | 92, 120, 0.77 | 92, Nothing happens, N/A |
| medium.txt | 25, 13, 1.9 | 25, 13, 1.9 | 25, 13, 1.9 | 25, 13, 1.9 |
| texts.tar | 1,350, 989, 1.4 | 1,350, 584, 2.3 | 1,350, 577, 2.34 | 1,350, 576, 2.34 |
| wacky.bmp | 901, 5, 180.2 | 901, 4, 225.3 | 901, 4, 225.3 | 901, 4, 225.3 |

winnt256.bmp 154, 156, 0.99 154, 62, 2.5 154, 62, 2.5 154, 62, 2.5

Using this chart of the sizes of various test files in the project before and after compression, as well as their ratios allows us to see the effectiveness and efficiency of the 4 methods of compression above. LZWmod.java with or without reset compresses the file size, for the most part, very well. I speculate that the compress.exe executable provided to us for the project is similar in implementation to LZWmod.java, because the compresses sizes and ratios are very similar for those two programs. For LZW.java, however, the compression was inefficient and ineffective. Compressing files with LZW.java not only took longer than LZWmod.java and compress.exe, but the compressed file sizes were not as small as the compressed file sizes in the other programs. The compression ratios for LZWmod.java and compress.exe were higher and provided great compression for bmp files.

Some files, like bogusInput.txt and bogusOutput.txt, are unable to compress because there was nothing in these files to compress. For example, an empty txt file cannot compress any further. In LZWmod.java and LZW.java, we throw an exception if this occurs, but in compress.exe, the file is just left alone and not compressed.

There were also some files that gave bad compression ratios. After compressing them, the file size could be potentially bigger. We saw this with frosty.jpg and gone\_fishing.bmp. z. I believe that the reason for this was that these files were already compressed before attempting to compress again, so that messed with the file size.

Overall, this LZW project was a great way to learn about compression methods. I’d always wondered how compression worked with large files, so it was great that I got to learn about that in a class like 1501. I’m very grateful for the opportunity to expand my knowledge on Computer Science.