|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Runtime to Compress (seconds)** | | | | |
| **File** | **LZW** | **Mod -n** | **Mod -r** | **Compress.exe** |
| All.tar | 2809 | 9.83598 | 29.72625 | 0.16 |
| Assig2.doc | 1.08 | 0.41533 | 0.41129 | 0.04 |
| Bmps.tar | 235 | 22.80613 | 22.42693 | 0.06 |
| Code.txt | 0.46 | 0.28421 | 0.29220 | 0.04 |
| Code2.txt | 0.34 | 0.26319 | 0.25317 | 0.04 |
| Edit.exe | 8.66 | 1.25493 | 1.27494 | 0.07 |
| Frosty.jpg | 3.57 | 1.40098 | 1.54408 | ---------- |
| Gone\_fishing.bmp | 0.15 | 0.19814 | 0.22515 | 0.04 |
| Large.txt | 202 | 3.28959 | 4.48817 | 0.1 |
| Lego-big.gif | 2.02 | 1.32394 | 1.15982 | ---------- |
| Medium.txt | 0.19 | 0.19314 | 0.19314 | 0.04 |
| Texts.tar | 405 | 3.76766 | 5.03869 | 0.13 |
| Wacky.bmp | 1.03 | 19.53921 | 20.86679 | 0.04 |
| Winnt256.bmp | 3.5 | 1.00871 | 0.92066 | 0.05 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Runtime to Decompress (seconds)** | | | | |
| **File** | **LZW** | **Mod -n** | **Mod -r** | **Compress.exe** |
| All.tar | 0.32 | 0.35025 | 0.47834 | 0.09 |
| Assig2.doc | 0.11 | 0.10909 | 0.10711 | 0.04 |
| Bmps.tar | 0.18 | 0.15409 | 0.15311 | 0.05 |
| Code.txt | 0.1 | 0.12014 | 0.11709 | 0.04 |
| Code2.txt | 0.11 | 0.10709 | 0.10409 | 0.04 |
| Edit.exe | 0.12 | 0.14411 | 0.14411 | 0.05 |
| Frosty.jpg | 0.12 | 0.13810 | 0.14016 | ----------- |
| Gone\_fishing.bmp | 0.1 | 0.10009 | 0.09807 | 0.03 |
| Large.txt | 0.19 | 0.23016 | 0.29220 | 0.06 |
| Lego-big.gif | 0.11 | 0.15311 | 0.13109 | ---------- |
| Medium.txt | 0.1 | 0.10407 | 0.10809 | 0.06 |
| Texts.tar | 0.22 | 0.22018 | 0.31522 | 0.08 |
| Wacky.bmp | 0.13 | 0.13110 | 0.12809 | 0.05 |
| Winnt256.bmp | 0.11 | 0.13810 | 0.12108 | 0.05 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Compression Ratio (original size / new size)** | | | | |
| **File** | **LZW** | **Mod -n** | **Mod -r** | **Compress.exe** |
| All.tar | 1.639344 | 1.666667 | 2.564103 | 2.564103 |
| Assig2.doc | 1.162791 | 2.173913 | 2.173913 | 2.173913 |
| Bmps.tar | 1.190476 | 12.5 | 12.5 | 14.28571 |
| Code.txt | 2.325581 | 2.857143 | 2.857143 | 2.941176 |
| Code2.txt | 2.380952 | 2.702703 | 2.702703 | 2.777778 |
| Edit.exe | 0.943396 | 1.515152 | 1.538462 | 1.5625 |
| Frosty.jpg | 0.714286 | 0.769231 | 0.740741 | --------- |
| Gone\_fishing.bmp | 1.851852 | 1.923077 | 1.923077 | 1.923077 |
| Large.txt | 2 | 2.380952 | 2.272727 | 2.325581 |
| Lego-big.gif | 0.724638 | 0.763359 | 0.763359 | --------- |
| Medium.txt | 1.923077 | 1.960784 | 1.960784 | 2 |
| Texts.tar | 1.369863 | 9.090909 | 2.325581 | 2.439024 |
| Wacky.bmp | 250 | 250 | 250 | 250 |
| Winnt256.bmp | 99.0099 | 2.5 | 2.5 | 2.5 |

* Note: Compress.exe refused to work on Frosty and Lego-big. I believe this is because their formats mean they were already more or less maximally compressed.
* Note II: I originally calculated the Compression ratio in the inverse way, and recorded them as percentages. As such, the data in each entry is now (prev / 100)^-1

Analysis of data section: It is clear that across the board, Compress.exe (or the linux terminal program ‘compress’) is vastly superior. This is likely because it is the standard implementation of an open source operating system (meaning that it, too, is open source). If there were any way to improve runtime, complexity, or compression ratio of compress, the Linux LZW method would know how to use it. I expect that it achieves such excellent compression ratios through heuristics, including normal file patterns that can be utilized (such as not even attempting to compress the jpg) or possibly by using an intelligent sensor to tell when to dump a table vs when you keep using the table (essentially the extra credit for this assignment).

With regards to my own program – I was fairly impressed as to how it held up against the given distribution. Admittedly, it’s not like I came up with the algorithm (I just implemented it), but for the most part it was able to keep pace with the Linux standard compression algo. Differences in the – r / - n implementations obviously stem from the systemic choice represented by each: ‘n’ signifies that you think there will be little change in the variation of the file down the line (long but consistent) whereas ‘r’ forces you to start over, but you get to keep learning (short memory, but better learning). While each has different practicality in different situations. It was fairly clear that the differences were more or less evened out except in cases that take advantage of it one way or another (looking at Texts and All.tar).

I would like to say that the author’s code, LZW, help up alright. However, even though it took ages to run (one would hope the author chose a tradeoff between time and compression), it still lagged behind even my own implementation in most cases. In some cases, it was ludicrously bad (bmps.tar). Given that it was slow, unwieldy, and used String.substring() about a million times throughout, I would recommend it only as a toy example, and not for any real uses. Also, needing to pipe files into the input? It made the project more difficult (mostly because I debug using StdOut), but also just arose a question of why anyone would do that. It seems to give less functionality than using a File Object, and a Scanner or Reader would surely be a better alternative. All around, just not my favorite code.