Huffman Analysis

* After compressing various files via Huffman and RunLength encoding, I recorded the size (in bits) of the encoded files. Here are the results:

|  |  |  |  |
| --- | --- | --- | --- |
| **File** | **Uncompressed** | **RunLength** | **Huffman (compression %)** |
| **mySample.txt** | 232 | 984 | 304 |
| **q32x48.bin** | 1,536 | 1,144 | 816 (47%) |
| **medtale.txt** | 45,808 | 185,520 | 24,616 (46%) |
| **genomeVirus.txt** | 50,025 | 223,704 | 14,048 (72%) |
| **mobydick.txt** | 9,708,952 | 39,407,928 | 5,505,424 (44%) |

* Decompressed files shared size with the original uncompressed files

As expected, Huffman had a far greater compression rate than RunLength for any file featuring natural language. RunLength only appeared to be an effective compression tool on the .bin file, while Huffman files were almost always smaller than the uncompressed file. The only exception was my own file which was intentionally small even uncompressed. Here Huffman encoding actually increased slightly the number of bits used.

Interestingly when looking at 2 similarly structured files with very different sized (medtale and mobydick), I found there compression rate via Huffman encoding to be very similar (46%, 44%). This indicates that the rate of compression for Huffman encoding is consistent when examining files with similar content.

* **Note**: When using Huffman to compress an already compressed file, no errors were encountered, and this file could even be completely decoded if I used the decompression method twice on the file. However the ‘double-compressed’ file was larger bitwise than singly compressed.