# Huffman encoding

# Part 3 Analysis

Assuming the clear.txt file contains the poem that starts with “Mary had a little lamb”, the *clear.txt* file size is 486 bytes. The *coded.txt* file size is almost 4x larger, at 2084 bytes.

The reason our coded.txt file is significantly larger than the original clear text file is that we are writing out the 1’s and 0’s to the coded.txt file as a stream of ***bytes*** as opposed to a stream of ***bits.*** Each ‘1’ and ‘0’ that we place into coded.txt takes up one byte of space (8 bits), and since all of the Huffman codes for the clear.txt characters are greater than 1 symbol long, of course the encoded text would take up more space than the original clear text when we store the 1’s and 0’s this way.

The more appropriate comparison is to consider how many bits each file uses (where a 0 or 1 Huffman encoding each require 1 bit of storage, and each ASCII character requires 8 bits=1 byte of storage). The original clear.txt file has 466 characters, each needing 8 bits of storage, for a total of 3728 bits. The *coded.txt* file contains just 2084 1’s and 0’s (each needing just 1 bit of storage space) which is roughly 40% fewer bits than those required to store *clear.txt.* It is THIS comparison that appropriately shows the compression capabilities of Huffman Encoding.

# Part 4 Analysis

In comparing sizes of files clear.txt and codedalt.txt, It is evident that the size of codedalt.txt is significantly lower than that of clear.txt.The reduction in file size is attributed to the efficient encoding process implemented in Part Four A. In clear.txt, each character is represented by a fixed-size ASCII code, typically 8 bits (1 byte), leading to potential redundancy. By utilizing a bit-level approach and storing the codes directly as sequences of 0s and 1s without converting them to ASCII characters, the file size is minimized. This method eliminates the need for a fixed-size representation for each character, resulting in a more concise storage format.