# Design Document for CS261 Assignment 2

## Huffman Encoding/Decoding Design Document

For Huffman, I did a canonical Huffman encoding and decoding. Where after I generate the bits for the chars in the document. I sorted them from the least bits to the most bits. After which, each of the existing codes are replaced with a new one of the same lengths, using the following algorithm:

* The first symbol in the list gets assigned a codeword which is the same length as the symbol's original codeword but all zeros. This will often be a single zero ('0').
* Each subsequent symbol is assigned the next binary number in sequence, ensuring that following codes are always higher in value.
* When you reach a longer codeword, then after incrementing, append zeros until the length of the new codeword is equal to the length of the old codeword. This can be thought of as a left shift.

e.g.

After generating the bits, default Huffman symbols

A = 11

B = 0

C = 101

D = 100

Then we sort them

B = 0

A = 11

C = 101

D = 100

Lastly, we apply the algorithm and generate the canonical Huffman symbols

B = 0

A = 10

C = 110

D = 111

For the transmitting of header, what I did was after we sort the symbols, we will send the difference of the bits. The starting symbol will be the length of itself, and the rest of the symbols are just the difference of length from the previous symbol.

e.g. Using the previous A, B, C, D where A is length 2, B is length 1, C is length 3, D is length 3. The header will be as such.

B 1 A 1 C 1 D 0

Where B is the starting and its length is 1. A is the next shortest bit, its difference of bit from B is 1. C is the next shortest bit, its difference of bit from A is 1. Lastly, D is the last symbol and its difference of bit from C is 0.

As for the Decoding part, using the header we can get the length of the bits each symbol has, and we can use the same algorithm as how we generate the canonical bits for encoding to get the bits.

## Arithmetic Encoding/Decoding Design Document

For Arithmetic Encoding, I used a short instead of a char for range as if we use char the frequency/probability table can only have 1/255 for each and all the symbols. As have anything higher than 1/255 will cause overflow. Also, the frequency/probability table will be weak if we use 1/255 for all the symbols. Therefore, I chose a short instead, allowing us to have frequency/probability higher than 1/255.

For the frequency/probability table, I found a statistic which provided a table based on frequency of letters in English text and the rest of the symbol will be set as probability/frequency 1.

For the header of the Arithmetic Encoding, I just send all the 255 + 1(EOF) probability/frequency as the header. As we are considering the whole 255 chars, we know that the 1st number will be the probability/frequency for char 1 and 2nd number will be for char 2 and so on.

When decoding, the table will be generated based on the header. We have 2 frequency/probability table, high and low table. Where high table is the frequency/probability table for the symbols and the low table is just the previous symbol’s frequency/probability.

## Sending and receiving of packets

In my current implementation, each packet is split into 500B of data. We will split encoded message and header into 500B of data. As we are using TCP, it is lossless, so we do not need to be afraid of packet dropping. However, we do not send all the packets once, we send one packet first and once we get an ack we will send the next packet. So, the sequence of the packets will not get messed up.