Sunday, May 9, 2021

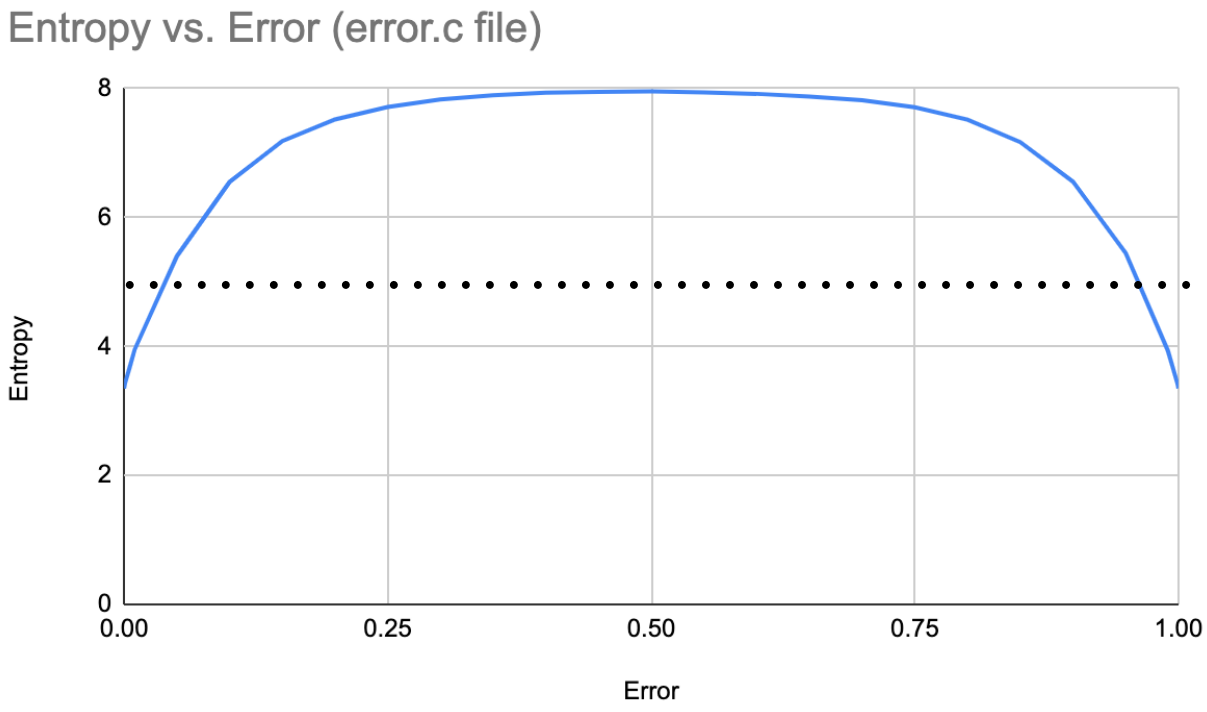
Assignment 5: Hamming Codes

Writeup

In this assignment, we implemented a Hamming Code (8, 4) encoder and decoder to ensure that bytes transferred in noisy environments that have errors can be recovered. Although this process is able to recover a lot of errors, there are still some bytes that have too many errors to be corrected. Those cannot be fixed with the (8, 4) Hamming encoder/decoder.

Entropy

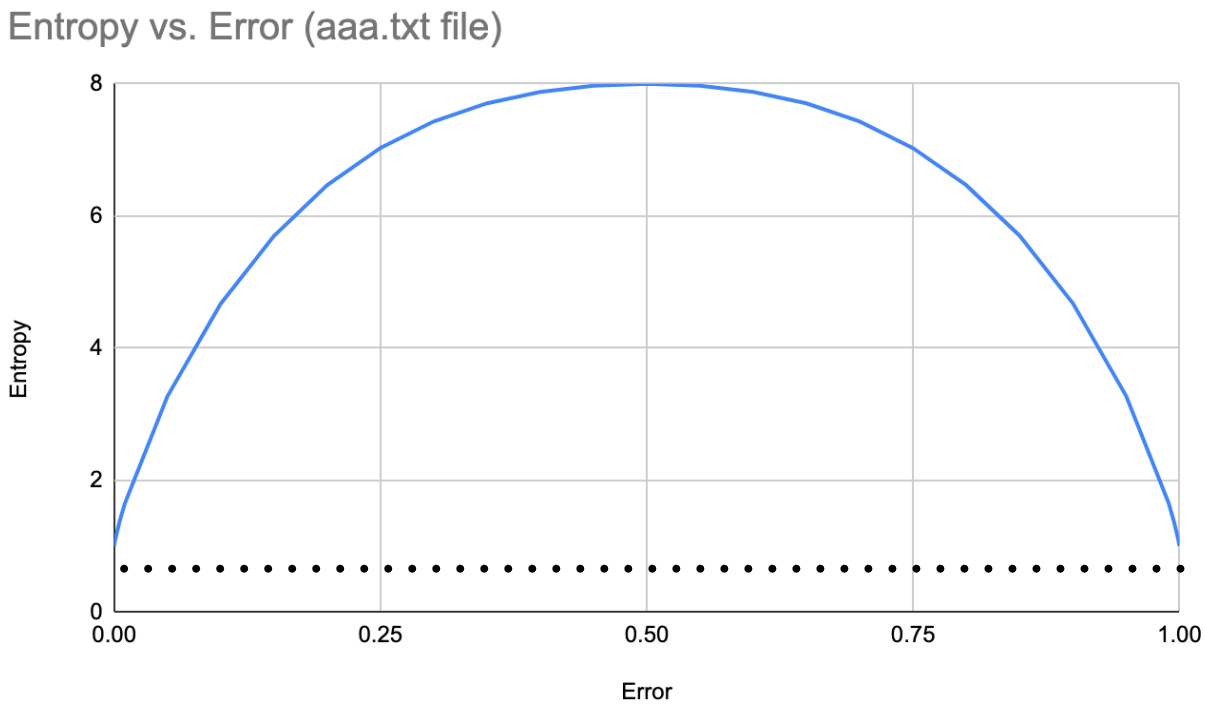
To test the entropy of Hamming codes, we use an entropy measuring program to measure the entropy vs error rate when we encode and decode with various injected error rates. Below is the resulting plot of my data points. This was tested on the text inside the error.c file.



From this graph we can see that the amount of entropy is greatest when we inject errors into the code at 50%. Before encoding and decoding, error.c had an

entropy of 4.685295, marked in the graph with a dotted black line. As the graph shows, the amount of entropy actually decreases after encoding and decoding when the injected error rate is extremely small or extremely large. A possible reason for this is that for this specific file, when encoding it and adding the extra characters, there is more repetition added and so less randomness in the encoded file.

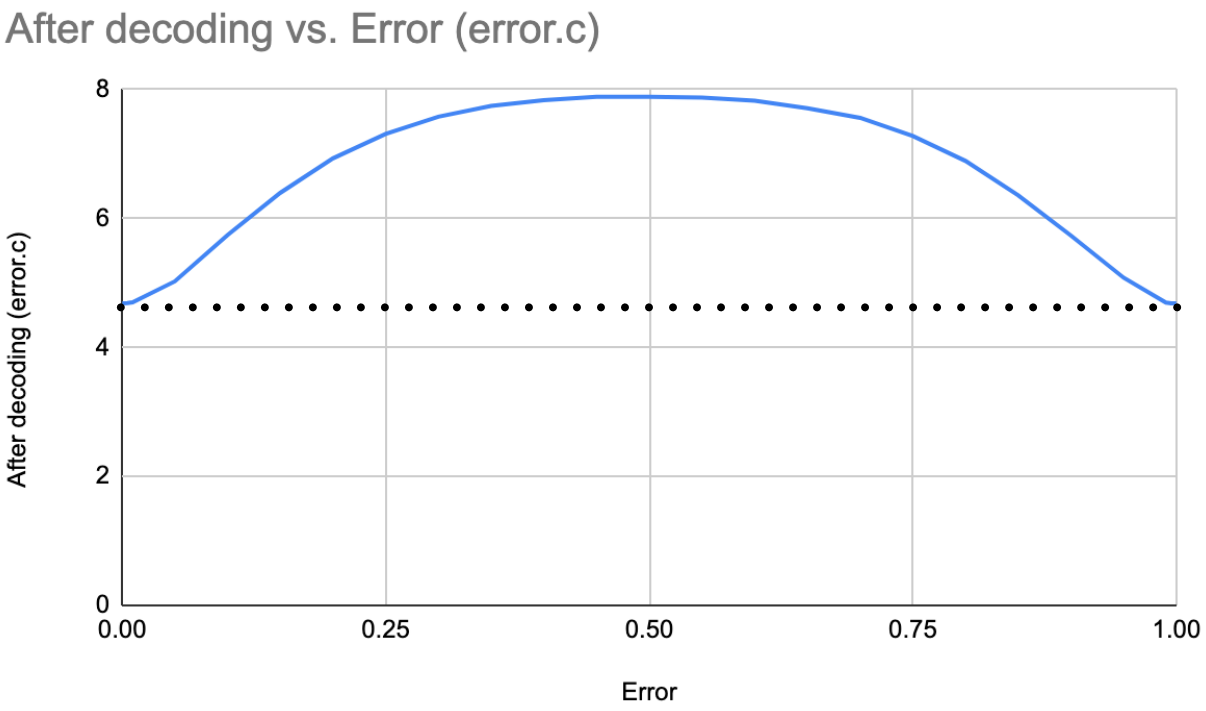
When more and more errors are added, there comes a point, 50%, when adding more errors will not contribute to the randomness, but will make the randomness slowly go back to what it was before injecting errors. After 50%, adding more error will just make the randomness of the file slowly return back to where it started (but the bits are slowly all becoming inverted). So at 100% error rate, the randomness of characters is the same as when it was 0% error, but every single bit will be flipped so they will be difference characters. This can be observed from the graph as it shows that the levels of randomness is mirrored on the axis of x = 0.5 (50%), where the max entropy, of almost 8, occurs.



I then performed the same tests but on a different file where the entropy of the file was very low, shown above. aaa.txt is a file filled with only the character a and has an entropy of 0 (no randomness) which means the entropy before encoding is 0, as

marked with the black dotted line. The entropy curve graph for this file has a similar shape to the previous one.

After encoding, the aaa.txt text has now become a repeating pattern of two characters. Which is more entropy than a single character repeating so that explains why the entropy after encoding is never less than before encoding. Same pattern as before, when the injected amount of errors reaches 50%, that is a tipping point where adding more errors will not increase the amount of randomness. It will decrease the randomness until eventually all the bits are flipped and randomness is the same as when there were no errors injected into the file.



For my final set of tests, I decided to compare the entropy between before encoding the text (black dotted line) and the entropy after encoding, injecting errors, then decoding (blue solid line). The shape of this curve is more bell shaped than the other two. We are comparing the entropy of the file after decoding to its original file untouched. At the beginning, the entropy after decoding is very close to the original file's entropy. This is because Hamming code (8, 4) can only correct 1 bit per byte of encoded data. So if a byte happens to have too many errors, it won't get fixed (properly) and at a low rate of error injection, it is more likely to have correctable errors.

And when there are more errors, the harder it is to correct and eventually the hamming code might be corrected into the wrong character. The rest of the pattern is the same as the above graphs, where the curve is mirrored across the x = 0.5 (50%) axis.

Overall, Hamming code is very useful tool that can be used to correct files in noisy channels. They are most effective in correcting files that have relatively low rates of error (1 bit per byte of encoded data).