Sam Weismuller
WES 237B
7/22/2022

Assignment 1 Report

Encoding scheme:
The Huffman tree information was saved to the coded file by saving the frequency counts for all 256 characters as unsigned integers. The Huffman tree can then be reconstructed from the frequency table the same was it was originally generated. Because we are saving 4 bytes for each of the 256 characters, the overhead of saving the encoding information comes out to 2 kB. This could be possibly improved by leaving out characters that weren't present in the file (assuming we are encoding plain text) or by using a smaller data type for the character frequency. However, I judged that 2 kB is reasonable enough given that the type of file we would get a tangible benefit from compressing would likely be several times that length.

The actual Huffman tree was built using a priority queue. Both character nodes and internal nodes were placed on the queue and their priority was determined by the sum of the frequencies of their child nodes (or themselves, for character nodes). The two lowest frequency nodes are pulled of the list and are used to build a new internal node. This process repeats until one internal node is remaining. This results in a tree that naturally balances symbol length (the depth on the tree) and character frequency.

Once the Huffman tree was generated, it was used to determine the encoded symbol for each character. This allows the encoding for each character to be quickly generated as we scan through the input file. On the decoding end, the tree itself was traversed using the incoming 0's and 1's to retrieve the character.

Note that a radix sort was used to do the initial sorting of characters by frequency. This wasn't entirely necessary as it would have been simpler just to use the priority queue to do the initial sorting, but it was mostly just an exercise to help me understand the algorithm.

Changes to the starter code:
No changes were made to main.cpp or the header files. All changes were made in huffman.cpp to implement the encoding/decoding scheme as described above. To account for the way main.cpp writes to the output text file, a null character was appended to the end of the decoded character buffer.

How to run the code:
The code can be build by running “make” from the main directory. The error about opencv can be ignored as that don't affect the compilation.

Sample output:
The files “code.txt” and “output.txt” in the repository were generated from running:
./CODEC input.txt code.txt output.txt

Note that while code.txt is a txt file, it actually contains binary data and can't really be read as text.