Max Chi
ECE 368 Project 2

Huffman Compression

For the ECE 368 project 2, the assignment had asked for the use of a Huffman Compression algorithm to take in an input text file and output a compressed version of it. This is done in a myriad of steps. First step is to count the frequencies of unique characters that appear in the text based on their different ASCII letters. From there, proceed to organize a frequency table and order them from the least frequent to most frequent order. After the characters have been properly ordered, they are assigned weight values, which are equivalent to their frequencies. After this, we start by merging each individual instances of characters together by combining the weights with the 2 least "weight" values respectively to create a node. This process is repeated until there is only 1 node left. At this point, an organized tree has been constructed, and it is necessary to log the paths to traverse in order to access the leaves, which house the characters. It is necessary to construct a codebook to house the respective paths taken in order to access the leaves of corresponding characters. The program will basically assign smaller bit values to represent repeated letters and larger bit values to less often recurring characters. This is done so that it will require less bits to access those characters that are more frequently used, effectively packing the same amount of information that was originally inputted into a lot less space by using bit operations. After compressing the input file, it is necessary to be able to decompress in order to further check if the compression process was successful. But it is also necessary to understand how the compression process works first.

COMPRESSION

In the compression process, the input file is first received and read into a function that counts and creates a list of frequencies for characters. A tree is then formed by merging the nodes that are assigned from the frequency list. The basic concept is taking the two lowest frequency tree nodes and joining them together, adding their weight together to make a parent node. This process continues until there is only the root left. When the tree is fully processed, each left branch is assigned a value of 0 and right branch a value of 1. This is to help map the character traversal paths. The codebook basically logs the paths of each unique character. The more frequent characters require less bits to travel, while less frequent characters will have more bits to access. The header is then written to the file and contains the binary tree in which the codebook has recorded, then a new line is inserted. Traversing the file and sort the letters in the file while referring to the codebook that was created using the tree. The file is then appended onto the

end of the header and saved with ".huff" at the end to signify that its been through compression.

DECOMPRESSION

In Decompression, the file is received and read until a new line is encountered, at which the header is located, the binary tree from which the codebook was created for the compression algorithm. The binary tree is reconstructed using the header. The the est of the file is now read and as it is read, the binary tree refers to each value's corresponding letter, and decodes the file with the help of the codebook. The file is then saved with a trailing ".unhuff" to show that it has been decompressed.

FORMAT OF FILE

The format of the compressed file is in binary format.

HUFFAMN TREE

A Huffman tree is used to create the codebook to compress the file and is used to build the tree to decompress the file. It is stored as the header in the compression file.

COMPRESSION RATIO

The compression ratio is the size of the original file divided by the size of the compressed file. An sample from a test file of 175 bytes compressed to 66 bytes has a compression ratio of 2.65.