Bryan Settles

ECE 368 Project 2 Report

In this assignment we were assigned the responsibility of making a Huffman compression algorithm in order to take an input file, compress it, and be able to decompress it and return it to the exact original input file. The correct way to do this is to measure the frequency with which the different ASCII characters appear in the given input file, to assign bit values to each of the characters that appear in the file, and to do it in a way that allows the characters to be accessed and decompressed in the correct order to return it to the exact original input file when it is being decompressed . The smaller bit values are reserved for the letters that are most often repeated, and the bigger bit values are given to the characters that appear the least often in the input file. This allows for more efficient space usage, and hence, the file can be compressed to a smaller size.

After I mapped the characters to their frequencies in the input file, I sought to create a binary tree, with the location of the letters in the tree based on the number of occurrences of that particular character. In my program, to create the tree, I joined two lowest frequency tree nodes together, adding their frequencies to make their parent node. I repeated this process for all of the different characters and frequencies until the tree was complete. After this, I created a binary sequence to assign to each character, giving the characters with the highest frequencies the shortest binary sequences possible. I created a number of helper functions in order to organize the process and to prevent any confusion or human error on my part. After that, the text file is then traversed and letters are stored according to the binary sequences that

they were previously assigned using the binary tree. At the end of this process, the filename is then changed to include the ".huff" that was required to come at the end of it.

For the program to decompress the encoded file, I immediately changed the filename to include the ".unhuff" to come after the ".huff" that was added earlier, and began decoding the message by reading the header, which then created the corresponding tree nodes and list nodes. I created several helper functions to assist in this process as well. When the newline character, '\n', is encountered, the header has been fully read, and the reconstruction of the binary tree begins. While the file is read, the binary tree is referenced so the output file can be correct, as well as to make sure that the output is put in correct order. The data is read in binary but converted to characters by referencing the Huffman tree. It goes until the end of the file is reached.

For file sizes of less than 1 KB the compressed file size was actually greater than the input file size. This shows that Huffman compression works best with large input files, and not for smaller ones. The time taken by huff.c and unhuff.c grows fast with increased input size.

Notes

No excess flags were used to compile this program.

As stated in the instructions, there is only 1 argument to the program, the name of the file to be encoded or decoded.

"./huff" is used to compress the file, and "./unhuff" is used to decompress the file.