Stephen Corya
ECE 368 Project 2 Report
Huffman Coding
10/31/14

For this project, I wrote an algorithm to read a file from the disk, run a Huffman encoding on the file, print the encoded file to the disk, and finally read and decode the file again. The decoded output files are identical to the input files according to the lack of output from the Unix “diff” command. This program does not work when the input file contains a ‘\n’ character. I attempted to debug this issue to no avail.

I used the same struct to construct the list of nodes used to build the Huffman code tree as I used to construct the tree itself. This saves memory and allows for a simple construction of the Huffman tree from the “forest” of single Huffman trees (in this case the forest is stored as a list). I kept this list in proper order by only adding nodes in the correct order.

The header for my output file is stored as a series of characters and integers. These represent the ASCII values and weights of those ASCII values in increasing ASCII order. This is the same order that is used to generate the Huffman tree in both the compression and decompression programs. This allows for consistency in the Huffman trees used to both encode and decode the file.

The output (compressed) files are actually larger than the input files in terms of bytes of storage in most cases. They are compressed in terms of the number of characters the contain. I attempted to optimize the output into binary, but I was not successful.

I used recursion to traverse the binary trees and lists in most case. In cases where values needed to be tracked, traversed these structures iteratively. I did not enjoy this project as well as I did Project 1. I found this project lengthy and encountered great difficulty managing my time as I worked on it.