ECE 368 Project 2
Michael Tam

Solution:

My compression function calls up and array and then increments the corresponding character value when they show up the in the file. It then takes the array and forms a priority queue with characters that show up more than once, weighing each one by the amount of times it shows up. The program then takes the queue and combines the nodes together to form a Huffman tree. The bit encodings are calculated by traversing this tree and are stored in a array, with a parallel array holding the corresponding characters. Finally, the contents of the file are pulled into a separate array and then the file is encoded character by character. The header contains the number of significant bits in the header and the file, as well as a preorder traversal of the tree.

The decompression function has an easier job: it pulls the data from the file, forms a Huffman tree from the given preorder traversal, and then traverses through the tree repeatedly until the file returns to as it was before the compression.

Data Structures used:

Huffman coding hinges on binary trees, so both programs implement it in some way. For the compression program, the tree is used primarily to calculate the shortened values that the characters will take. In the decompression program, a tree is built so that following a path down the tree leads to the corresponding original character that the shortened characters in the compressed files present. Aside from that, the compression program uses a priority queue to build the Huffman tree, as frequent characters would benefit from a shorter encoding.