Asheem Chhetri
00270-84934

Project 2 milestone 1

The purpose of the project is using queues and trees, to create two programs: "huff.c" and "unhuff.c". The program "huff.c" accepts one argument that is name of the input file and compresses the input file using Huffman coding, and stores the compressed output to a file with same name as argument with ".huff" extension. The program "unhuff.c" accepts one argument, which will be the name of an input file. The program de-compresses the input file and write out the de-compressed output to a file that has the same name as the input file with an ".unhuff" extension. **Note:** We will be making an assumption that the, input file to "huff.c" will be a pure ASCII text file.

We can do conversion from ASCII to binary using their ASCII code to help us with assignment. We can further use only 3-bits to encode the 8 different characters. For creating a tree, a binary tree will be used such that all characters are stored at the leaves of a tree. We need to be specific on the tree structure, as coding varies for different tree. The stream will be decoded, by starting at the root of the tree, and follow a left-branch if the next bit in the stream is a 0, and a right branch if the next bit in the stream is a 1. When a leaf is reached, write the character stored at the leaf, and start again at the top of the tree.

Thus to compress a file (sequence of characters) we will need a table of bit encodings, i.e., a table giving a sequence of bits used to encode each character. This table is constructed from a coding tree using root-to-leaf paths to generate the bit sequence that encodes each character. We then:

- Build a table of per-character encodings
- Read the file to be compressed (the plain file) and process one character at a time.

To compress the string/file we read one character at a time and write the **sequence of bits** (i.e. not 0/1 characters) that encodes each character.

We then build a table of optimal per-character bit sequences for which we need to build a Huffman coding tree using the Huffman algorithm. The table is generated by following every root-to-leaf path of the Huffman tree and recording the 0/1 edges followed. These paths represent the optimal encoding bit sequences for each character.

There are three steps in creating the table:

1. Count the number of times every character occurs. Use these counts to create an initial forest of one-node trees. Each node has a character and a weight equal to the number of times the character occurs.
2. Use the greedy Huffman algorithm to build a single tree. The final tree will be used in the next step.
3. Follow every root-to-leaf path of the tree creating a table of bit sequence encodings for every character/leaf.

For the header information, we need to store some initial information in the compressed file that will be used by the unhuffing program. We must store the tree that is used to compress the original file. This is because the decompression program needs this exact same tree in order to decode the data.

For the unhuff program we cannot simply read bits until there are no more left since the program might then read the extra padding bits written due to buffering. To avoid this problem,

we need to use a pseudo-EOF character and write a loop that stops when the pseudo-EOF character is read in (in compressed form).