# Project 2 Milestone 1

By Glenn Klahre(gklahre)

First thing required for Huffman coding is to find out exactly how many of each character/symbol are in the text. So a pass will be done over the entire text, counting the number of times each one comes up. This will be done in an integer array, with each character being represented by its ascii number. In other words, 'a' is 97 in ascii, so array[97] would give the number of times 'a' was used.

Now the Huffman tree must be created. The leaf nodes of the Huffman tree will be created, keeping them in a sorted list (smallest to largest) as they are being made. A Pseudo-EOF character will be added with a count of 1. Then, the tree will be created, as described in the p2 assignment text: The smallest two will be combined, by connection to an in inner node. Their counts will be added, and the node will be added to the list, as the two leaf nodes used will be removed.

A file will be created using the input name given, and adding the required '.huff'.

The header will be created using a preorder system. 0 for non-leaf node followed by its character in ascii, 1 for leaf node. In order to make the compression the smallest possible, bitwise operators and masks will be used. This starts to get a bit complicated. Whenever space has run out in a bit, it should be printed and a new bit should be created to start work on. The exact codes for each character will be stored in a 2D array. This will be done using tree traversal, a secondary integer string array, and a depth tracker. As the traversal goes deeper into the tree, the depth tracker will change which integer in the string gets changed. It also notifies how many digits should be included in string that will be put into the 2D array. This be for the encoding process.

With the position array created and header completed, encoding the actual text can begin. Each letter can now be encoded using simple 1's and 0's. As things go, this is the simple part. The Pseudo-EOF character will be added once all other characters are encoded.

Decoding will simply involve creating a tree using the header, and traversing the tree for the rest of the text. Once the Pseudo-EOF comes up, the decoding with stop. The file be will be labeled with the '.unhuff'.

I will be looking at using lists rather than arrays in some parts of the code for efficiency sake, but that will come after I get a complete working code going. For instance, if the text only has the letters ‘I’ and ‘E’ then having a complete array to sort will be unnecessarily time consuming.