Huffman Codes

How Does it Work?

Huffman codes work by utilizing a binary tree correlating its frequency to its height in the tree. The more frequent, the closer to the root it will be. A Huffman encoding algorithm takes the two least frequent characters/symbols/representation of characters and attaches them to a parent node with frequency of both previous nodes added to them. The encoding algorithm continues until one item is left in the queue, that being the tree.

Once the tree is created, traversing left will yield a 0 and right a 1. If you want to decrypt the file, follow the bits in the encoded document. If you want to encrypt, create a mapping of the characters to their correlated location in the binary tree (the bit string).

My Encoding

Pair Huffman Encoding:
{' m': '0000000', 'es': '0000001', 'wa': '0000010', 'd ': '0000011', 'ss': '0000100', '!': '0000101', 'ha': '0000110', 'y ': '0000111', ':': '00010000', 'O': '00010001', 'U': '0001001', 'la': '0001010', '0': '0001011', 'p': '000110', 'ar': '0001110', 'ut': '0001111', ' ': '001', 'r': '01000', 'e ': '010010', 'ou': '010011', 'g': '010100', 'T': '010101', 'c': '010110', 'F': '0101110', 'is': '0101111', ' t': '011000', 'u': '011001', 'o ': '0110100', ' I': '0110101', '. ': '0110110', 'w': '0110111', 'm': '011100', 'W': '011101', 're': '0111100', 'to': '0111101', 'it': '0111110', 'as': '0111111', '.': '100000', '"': '1000010', 'm ': '1000011', 'I': '1000100', 'C': '1000101', 't ': '100011', 'a': '10010', 'i': '100110', 'l': '100111', 's ': '101000', 'n': '101001', 't': '10101', 's': '10110', 'h': '101110', ' c': '10111100', 'v': '101111010', '2': '101111011', 'ne': '10111110', 'yo': '10111111', 'o': '11000', 'H': '110010000', '{': '110010001', ' w': '11001001', 'me': '11001010', 'CT': '11001011', 'L': '110011000', 'Z': '110011001', 'le': '11001101', 'co': '11001110', '..': '11001111', 'k': '110100000', '7': '110100001', 'on': '11010001', ' o': '11010010', 'ab': '11010011', 'N': '11010100', 'A': '11010101', '}': '110101100', 'M': '110101101', '6': '11010111', 'er': '11011000', 'TF': '11011001', 'd': '1101101', 'th': '1101110', "I'": '11011110', "'m": '11011111', ' i': '1110000', 'b': '1110001', 'J': '11100100', '8': '11100101', '3': '111001100', 'E': '111001101', 'ur': '11100111', 'e': '11101', 'y': '111100', ' a': '1111010', 'hi': '11110110', 'S': '11110111', ' y': '11111000', 'f': '11111001', '-': '111110100', 'K': '111110101', 'x': '111110110', '?': '111110111', 'e.': '11111100', ' s': '11111101', ' h': '11111110', 'ag': '11111111'}

People *should* have identical mapping to what's listed. If they don't, there shouldn't be a way to use the bits as directions in the document AND receive the correct answer.

Troubleshooting

If someone is getting nearly correct output, ensure every character or character pair is somewhere in the document. For example, if there is no sequence of ‘111001100', we know it is wrong because that string represents 3. 3 does appear in the document, so this is something to note.