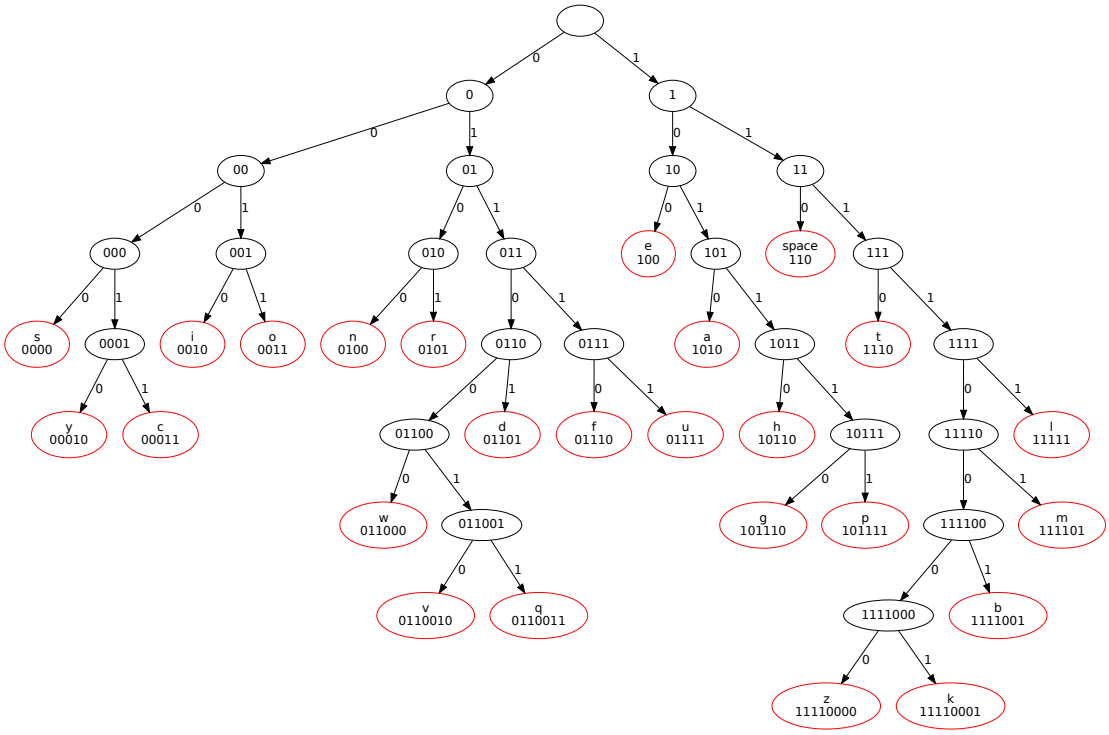
Day 4: Squeeze until the bits squeak!

Your droid P0P0 is in bad shape. While debugging his encryption algorithms, you accidentally caused a short circuit and destroyed all of his persistent memory chips except for 24 bytes.

Before shutting down, P0P0 transferred some data into the remaining memory. It must be a very important message that he needed to preserve.

In order to fit the message into just 24 bytes P0P0 needed to compress the data by reducing the number of bits required to store each character. Being a clever droid, he used an algorithm to make the most frequently used letters require the smallest number of bits. For example, the letter ‘e’ (used in 13% of letters in English) is represented with only 3 bits, 100. The letter ‘z’ (hardly used, 0.7%) is represented with 8 bits, 11110000.

With a final puff of smoke, P0P0 prints out a tree showing how he has compressed the data:



This is what you find in P0P0’s 24 bytes of working memory:

10110100 11111101 11111011 11011001 10001111 11001001 01100110 00001101 00110111 10001100 01000011 11110010 01011000 01000110 11111101 01001011 00110111 10100010 11000110 10011111 00010110 10110001 11011111 00110110

- What is the very important message?
- What fraction of the size is the compressed message when compared to the original ASCII text? (This is called the “compression ratio”.)
- **Bonus marks:** How would you create a tree for a different language with different character frequencies? (Google “Huffman Encoding”, for example the Wikipedia page.)