

Definition We say that a binary code has the **prefix property** if a code symbol never appears as an initial segment (that is, as a prefix) of another code symbol.

The weight of a code

The point about variable length codes is that they can be economical when some characters appear with far greater frequencies other than characters. In order to construct a variable length character code with the prefix property we ought therefore to take into account the frequencies with which the characters appear in a given context (such as letters of the alphabet in English text).

We will use a finite character set: $\{C_1, C_2, C_3, \dots, C_n\}$ with frequencies $f_1, f_2, f_3, \dots, f_n$ respectively. The frequencies $f_1, f_2, f_3, \dots, f_n$ are positive numbers of a nature that is determined by the context: most commonly, they are natural numbers, or they are decimals in the range $[0,1]$.

We want a measure of how economical a given variable length code is for the character set $\{C_1, C_2, C_3, \dots, C_n\}$. The measure we use is the weight of the code. This is obtained by multiplying the frequency f_i of the character C_i by the length of the code symbol for C_i , and then summing from 1 to n . For example, suppose the character set is $\{a, b, c, d, e, f\}$ with frequencies given by the following table:

| Character | a | b | c | d | e | f |
|-----------|------|------|------|------|------|------|
| Frequency | 0.44 | 0.26 | 0.14 | 0.09 | 0.06 | 0.01 |

Listed below are two variable length character codes for this character set, and both codes have the prefix property:

| Character | Code symbol (1st code) | Code symbol (2nd code) |
|-----------|------------------------|------------------------|
| a | 000 | 0 |
| b | 001 | 11 |
| c | 11 | 101 |
| d | 10 | 1001 |
| e | 010 | 10001 |
| f | 011 | 10000 |

The weight of the first code is:

$$(0.44 \times 3) + (0.26 \times 3) + (0.14 \times 2) + (0.09 \times 2) + (0.06 \times 3) + (0.01 \times 3) = 2.77.$$

The weight of the second code is:

$$(0.44 \times 1) + (0.26 \times 2) + (0.14 \times 3) + (0.09 \times 4) + (0.06 \times 5) + (0.01 \times 5) = 2.09.$$

On average, the second code requires fewer bits to encode a given character than does the first code: about 25% fewer. The reason for this is that the second code uses short code symbols for frequently occurring characters and longer code symbols for rarely occurring characters.

Huffman codes

For a given character set, with given frequencies for the characters in a specified context, how can we construct a character code that is of at least weight among the variable length character codes with the prefix property?

One answer is given by a construction of binary trees, called **Huffman trees**. The corresponding codes are known as **Huffman codes**. We illustrate this procedure with the character set $\{e, g, i, n, r, s\}$ and the frequencies: 0.311, 0.046, 0.174, 0.167, 0.144, 0.158.

At each stage of the construction we have an ordered list of binary trees. The roots of the binary trees are labelled with the characters or numbers (or both), and the roots are arranged left to right in increasing order of numerical label (with an arbitrary arrangement among roots labelled with the same numerical label). There are 6 steps to the algorithm:

The Huffman Algorithm

Step 1. Create a root of a binary tree for each character. Label the roots in a row, in increasing order of frequency, from left to right. List the frequencies together with the characters. (If two characters have the same frequency then choose arbitrarily which will precede the other.)

Step 2. Replace the two leftmost roots with a single new root. Label this root with the sum of the frequencies of the roots it replaced. Create a binary tree with the new root, with the leftmost of the replaced roots as left vertex, and the rightmost of the replaced roots as right vertex.

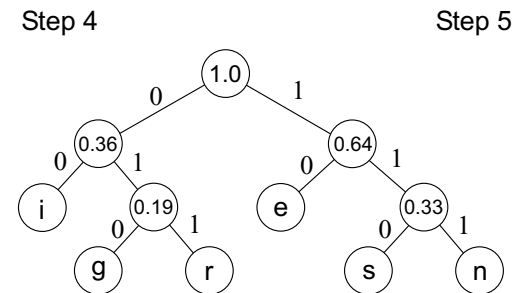
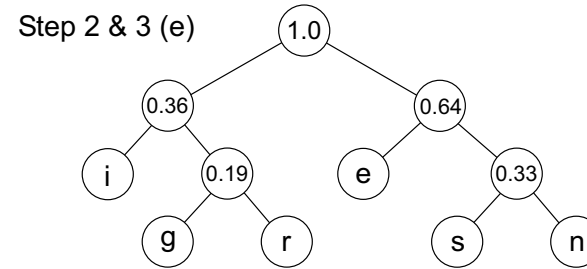
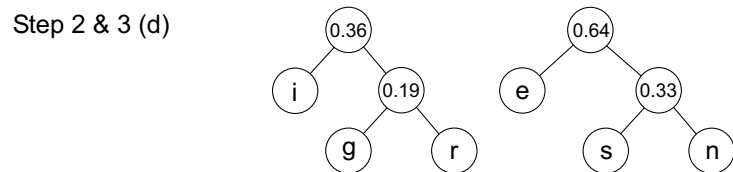
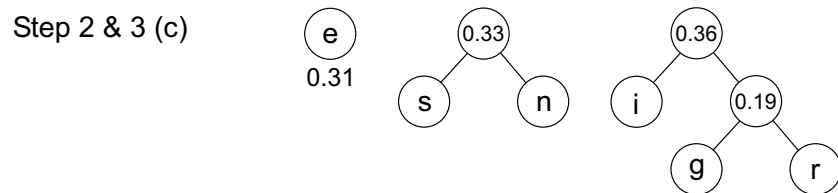
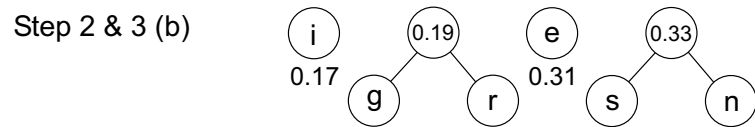
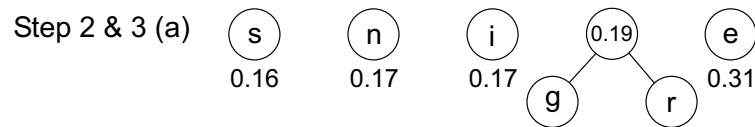
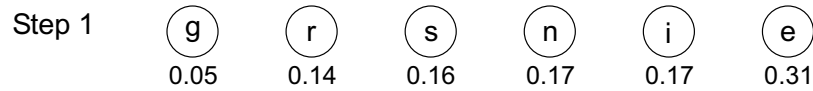
Step 3. Re-order the set of roots according to the numerical labels.

Repeat steps 2. and 3. until there is only one root.

Step 4. Label each left edge of the resulting tree with a "0" and label each right edge with a "1".

Step 5. For each character read the sequence of edge labels as you traverse from the root to the vertex containing that character. (This is the Huffman Code for that character.)

Example



The resulting tree is a Huffman tree for the character set and specified frequencies. A Huffman code is constructed from this tree in the following way. From step 5 of the algorithm every left branching edge of the Huffman tree is labelled with a 0, and every right branching edge with a 1. Notice that the characters are labels of terminal vertices of the tree. The code symbol of a character is the sequence of edge labels in an edge path from the root to that character (step 5.).

The weight of this Huffman code is:
 $(0.311 \times 2) + (0.046 \times 3) + (0.174 \times 2) + (0.167 \times 3) + (0.144 \times 3) + (0.158 \times 3) = 2.515$.

It's an important fact that the weight of a Huffman code is the least among weights of variable length character codes with the prefix property.

This tells us (because the frequencies summed to 1 in this example) that an average encoding rate of 2.515 bits per symbol is the best we can do if we want to code the character set $\{e, g, i, n, r, s\}$, with the given frequencies, by a variable length code with the prefix property

Encode "green"

Decode: 10111010001111010011110