9. Trees :: Applications :: Huffman Encoding (continued)

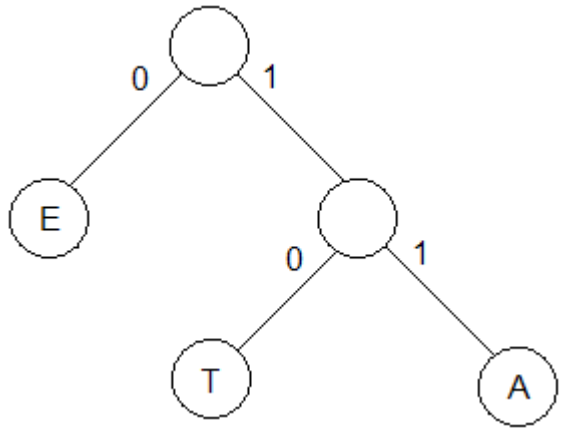
However, we would quickly discover there is a problem with this encoding. Suppose on the receiving end we receive the bits 0101; what sequence of letters does this represent?

```
0 1 0 1     01 01     01 0 1     0 1 01
e t e t     a  a      a  e t     e t a
```

Assuming etet, aa, aet, and eta were all English words, then we could not be sure which word we just received.

One way around this problem is to assign a bit pattern for each letter so the bit pattern for that letter never occurs as the first part of the bit pattern for another letter. This would mean that *a* could not be encoded as 01 because the bit pattern assigned to *e* (0) occurs as the first part of the bit pattern for *a*.

Symbolic binary encodings which conform to this rule are known as **prefix codes**. For example, suppose we encode *e* as 0, *t* as 10, and *a* as 11 and construct this binary tree:



9. Trees :: Applications :: Huffman Encoding (continued)

Then upon receiving 01110 we would decode this by starting at the root node:

1. The first bit is 0. Move to the left child. Since this is a leaf node, we know the first received letter is *e*.
2. Move back to the root note. The next bit is 1. Move to the right child of the root node. This is an interior node, so examine the next bit.
3. The next bit is 1. Move to the right child. Since this is a leaf node, we know the second received letter is *a*.
4. Move back to the root note. The next bit is 1. Move to the right child of the root node. This is an interior node, so examine the next bit.
5. The next bit is 0. Move to the left child. Since this is a leaf node, we know the third received letter is *t* and the received word is *eat*.

The most well-known prefix code is the **Huffman Code** and using Huffman's algorithm with the aforementioned letter frequency analysis of various English books, we obtain these prefix codes for the letters a–z (where the number in parentheses is the number of bits):

a = 0010 (4)	f = 000001 (6)	k = 0110110 (7)	p = 011001 (6)	u = 11000 (5)	z = 0110111 (7)
b = 011010 (6)	g = 000110 (6)	l = 00010 (5)	q = 0110000 (7)	v = 0110001 (7)	
c = 000000 (6)	h = 0111 (4)	m = 11010 (5)	r = 1011 (4)	w = 11011 (5)	
d = 00001 (5)	i = 0101 (4)	n = 0100 (4)	s = 1010 (4)	x = 110011 (6)	
e = 100 (3)	j = 110010 (6)	o = 0011 (4)	t = 111 (3)	y = 000111 (6)	

9. Trees :: Applications :: Huffman Encoding (continued)

From these encodings, we could generate the **Huffman tree** which would be used to decode the received bits (we will leave the drawing of the resulting binary tree as an exercise for the student.).

If we used these letter encodings for our sentence:

> however in english certain letters occur in words and sentences with greater frequency than other letters

The the total number of transmitted bits would be 368 as opposed to 450, which is a savings of 18.2%.

To conclude, let's mention that Huffman encoding can also used as a file compression method, e.g., Huffman coding is employed in compressing JPEG images and MP3 files.