Huffman Codes

Suppose we want to encode text to transmit across a communication channel (noiseless) and we want to transmit as few bits as possible.

Trivial approach: 27 symbols to represent, need 5 bits to cover 27 symbols, give each symbol a different 5-bit number as its encoding.

Is this an efficient approach? Why or why not?

Should take frequencies of different letters into account, give shorter encodings to more frequently occurring letters and longer encodings to less frequent letters.

Example—suppose we only have 7 letters plus space with following frequencies:

- space - 25%
- c - 7%
- e - 25%
- k - 5%
- a - 15%
- x - 2%
- t - 20%
- z - 1%
With basic encoding, each character uses 3 bits
and $E[\text{char. length}] = 3$

Transmit 1,000,000 characters $\Rightarrow$ use 3,000,000 bits

With distribution as given

Instead, consider the following encoding:

- Space: 11
- t: 10
- e: 01
- a: 001
- C: 0001
- K: 00001
- Z: 000001
- X: 000000

$E[\text{char. length}] = 2 \times (0.25 + 0.25 + 0.2) + (3 \times 0.15) + (4 \times 0.07) + (5 \times 0.05) + 6 \times (0.02 + 0.01)$

$= 2.56$

Transmit 1,000,000 characters $\Rightarrow$ use 2,560,000 bits

With given distribution

Savings of 440,000 bits!!

(Aside: What is an alternative approach to saving bits? Compression - discuss and compare with Huffman codes - compression not as efficient but can use standard off-the-shelf software and applies to any type of information)
the encoding I gave has a very special property — anyone see what it is?

no code word is a prefix of any other code word — explain this

why is this property important??

since this is a variable length code, the number of bits in a code word is not known beforehand — need this property to know when a code word ends

suppose we had $e: 01$, $a: 011$

if we see $0110...$

we don't know whether this is $0110...$

or $01100...$

however, w/ stated property, the end of each code word is unambiguous (because it is not a prefix for any other code word)

the given encoding was constructed w/ a Huffman tree and is referred to as a Huffman code
Huffman codes are probably the most efficient codes (shortest expected length) with the property that no code word is the prefix of any other.

**Constructing a Huffman Code**

1) take 2 least frequent symbols - assign their last bits to be 0 and 1, respectively
2) combine these 2 symbols into 1 symbol and add their frequencies together to get the frequency of this new logical symbol
3) repeat Step 1 and continue until all symbols are combined

4) for each symbol, follow bits from top of tree to symbol to get its encoding

- Space: 11
- a: 001
- z: 000001
- t: 10
- e: 01
- k: 0000
- x: 000000
Should be clear from the way the tree is constructed why no code word is a prefix of any other.

**Calculating Pi using Random Numbers**

![Diagram of a square and a circle with a dot inside]

Area of circle = \( \pi r^2 = \pi \)

Area of square = \( 2 \times 2 = 4 \)

Throw \( n \) darts randomly at square

\[ P \left[ \text{a dart lands in circle} \right] = \frac{\text{area of circle}}{\text{area of square}} \]

\[ = \frac{\pi}{4} \]

Suppose you throw \( n = 1,000,000 \) darts at square and \( x \) of them land in circle.
then, we estimate that

\[ \frac{\pi}{4} = \frac{2}{n} \Rightarrow \pi = \frac{4n}{2} \]

we can do this w/ a program that generates random points in the unit square (random x value in [-1,1], random y value in [-1,1]) and calculates how many are inside unit circle (dist. from [0,0] ≤ 1)

(show random Pi estimator program in Eclipse)

this is an example of Monte Carlo simulation, which is the use of random experiments to numerically estimate values