A wordle – a word collage.

A wordle constructed out of one of the instructor’s papers:

wordle constructed using the java applet at wordle.net
wordle uses a randomized greedy algorithm to solve the packing problem.

4.8 Huffman Codes

Encoding Text

Q. Given a text that uses 32 symbols (26 different letters, space, and punctuation characters), how can we encode this text in bits?

A. We can encode 2^5 different symbols using a fixed length of 5 bits per symbol. This is called fixed length encoding.

Q. Some symbols (e, t, a, o, i, n) are used far more often than others. How can we use this to reduce our encoding?

A. Encode these characters with fewer bits, and the others with more bits.

Q. How do we know when the next symbol begins?

A. Use a separation symbol (like the pause in Morse), or make sure that there is no ambiguity by ensuring that no code is a prefix of another one.

Ex. c(a) = 01                  What is 0101?
c(b) = 010

Prefix Codes

Definition. A prefix code for a set S is a function c that maps each x ∈ S to Is and Os in such a way that for x,y ∈ S, x≠y, c(x) is not a prefix of c(y).

Ex. c(a) = 11
c(e) = 01

c(k) = 001

c(l) = 10

c(u) = 000

Q. What is the meaning of 1001000001?
Fixed vs. Variable encoding

<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
<th>c</th>
<th>d</th>
<th>e</th>
<th>f</th>
</tr>
</thead>
<tbody>
<tr>
<td>frequency</td>
<td>45</td>
<td>13</td>
<td>12</td>
<td>16</td>
<td>9</td>
</tr>
<tr>
<td>fixed encoding</td>
<td>000</td>
<td>001</td>
<td>010</td>
<td>011</td>
<td>100</td>
</tr>
<tr>
<td>variable encoding</td>
<td>0</td>
<td>101</td>
<td>100</td>
<td>111</td>
<td>1101</td>
</tr>
</tbody>
</table>

100,000 characters
Fixed: 300,000 bits
Variable?

\[(1 \times 45 + 3 \times 13 + 3 \times 12 + 3 \times 16 + 4 \times 9 + 4 \times 5) \times 1000 = 224,000 \text{ bits}\]

25% saving

Optimal Prefix Codes

**Definition.** The average bits per letter:

\[ABL(c) = \sum_{x \in S} f_x \cdot c(x)\]

\[f_x = \text{frequency of letter } x\]

\[S = \text{alphabet}\]

We would like to find a prefix code that has the lowest possible average bits per letter.

Representing Prefix Codes using Binary Trees

Nodes labeled with characters
Path to a node represents its code

Ex. \(c(a) = 11\)
\(c(e) = 01\)
\(c(k) = 001\)
\(c(l) = 10\)
\(c(u) = 000\)

Q. What's special about the tree of a prefix code?

A. Only the leaves are labeled.

\[ABL(T) = \sum_{x \in S} f_x \cdot \text{depth}_T(x)\]

\[\text{depth}_T(x) - \text{the depth where } x \text{ occurs}\]

Q. How can this prefix code be made more efficient?
Representing Prefix Codes using Binary Trees

**Q.** How can this prefix code be made more efficient?
**A.** Change encoding of p and s to a shorter one. This tree is now full.

**Definition.** A tree is full if every node that is not a leaf has two children.

**Claim.** The binary tree corresponding to the optimal prefix code is full.

**Pf.** (by contradiction)
- Suppose T is binary tree of optimal prefix code and is not full.
- This means there is a node u with only one child v.
- Case 1: u is the root; delete u and use v as the root.
- Case 2: u is not the root
  - let w be the parent of u
  - delete u and make v be a child of w in place of u
- In both cases the number of bits needed to encode any leaf in the subtree of v is decreased. The rest of the tree is not affected.
- Clearly this new tree T' has a smaller ABL than T. Contradiction.

Optimal Prefix Codes: False Start

**Greedy approach.** Create tree top-down, split S into two sets S₁ and S₂ with (almost) equal frequencies. Recursively build tree for S₁ and S₂. [Shannon-Fano, 1949]

Example: \( f_a=0.32, f_e=0.25, f_k=0.20, f_l=0.18, f_u=0.05 \)

Can we do better?

Optimal Prefix Codes

**Q.** Where in the tree of an optimal prefix code should low frequency letters be placed?
Optimal Prefix Codes

Observation. Lowest frequency items should be at the lowest level in tree of optimal prefix code.

Observation. For more than two letters, the lowest level always contains at least two leaves.

Claim. There is an optimal prefix code with tree $T^*$ where the two lowest-frequency letters are assigned to leaves that are siblings in $T^*$.

Huffman encoding. [Huffman, 1952] Create tree bottom-up:

- Make two leaves for two lowest-frequency letters $y$ and $z$.
- Recursively build tree for the rest using a meta-letter for $yz$.

Example: $f_a=0.32$, $f_e=0.25$, $f_k=0.20$, $f_l=0.18$, $f_u=0.05$

Huffman(S) {
  if $|S|=2$
    return tree with root and 2 leaves
  else {
    let $y$ and $z$ be lowest-frequency letters in $S$
    $S' = S$
    remove $y$ and $z$ from $S'$
    insert new letter $\omega$ in $S'$ with $f_\omega = f_y + f_z$
    $T' = \text{Huffman}(S')$
    $T = \text{add two children } y \text{ and } z \text{ to leaf } \omega \text{ from } T'$
    return $T$
  }
}

Q. What is the running time?

A. Simple approach: $T(n) = T(n-1) + O(n)$ so $O(n^2)$

Using priority queue for $S$: $T(n) = T(n-1) + O(\log n)$ so $O(n \log n)$
Huffman Encoding: Optimality

Claim. The Huffman code for \( S \) achieves the minimum ABL of any prefix code.

The change in ABL when moving from \( T \) to \( T' \) (y and z removed, \( \omega \) added):

\[
ABL(T') = ABL(T) - f_\omega
\]

Claim. The Huffman code for \( S \) achieves the minimum ABL of any prefix code.

Proof. (by induction over \( n=|S| \))

Base: For \( n=2 \) there is no shorter code than root and two leaves.

Induction Hypothesis: Suppose Huffman tree \( T' \) for \( S' \) with \( \omega \) instead of y and z is optimal. (IH)

Induction Step:
- Let \( T \) be the tree generated by Huffman for \( S \).
- Consider another tree \( Z \) for which leaves y and z exist that are siblings and have the lowest frequency (see observation).
- Let \( Z' \) be \( Z \) with y and z deleted, and their former parent labeled \( \omega \).
- Similarly, \( T' \) is derived from \( T \).
- We know that \( ABL(Z') = ABL(Z) - f_\omega \), as well as \( ABL(T') = ABL(T) - f_\omega \).
- From our induction hypothesis \( ABL(T') = ABL(Z') \), so \( ABL(T) = ABL(Z) \).