**Huffman Coding**

How can we represent a string of characters with a minimum number of bits?

Answered by David Huffman in 1952 when he was a graduate student at MIT.

- Each character represented by a bit string code.
- Codes are of various lengths.
- Codes are prefix codes (prefix-free). No code is prefix of another.

\[
\begin{array}{ccc}
  a & b & c \\
  0 & 01 & 101 \\
\end{array}
\]

**Prefix codes**

\[
\begin{array}{ccc}
  a & b & c \\
  1 & 01 & 001 \\
\end{array}
\]

Not prefix codes:

\[
\begin{array}{ccc}
  a & b & c \\
  0 & 10 & 101 \\
\end{array}
\]
How would you represent a string of characters G, C, T, and A?

Assign shortest codes to most frequent characters. Let's say frequencies are

\[
\begin{array}{cccc}
G & C & T & A \\
0.4 & 0.3 & 0.2 & 0.1
\end{array}
\]

One prefix code is

\[
\begin{array}{cccc}
G & C & T & A \\
0 & 10 & 110 & 111
\end{array}
\]

The string

C G T T A C C A T A A A

is

10 0 110 110 111 10 10 111 110 111 111

Can you decode this? This binary tree helps:

\[ \text{Diagram of binary tree} \]
How many bits are needed for 1,000 G, C, T, A's?

\[
\begin{array}{cccc}
G & C & T & A \\
0.4 & 0.3 & 0.2 & 0.1 \\
\end{array}
\]

\[
\begin{align*}
G &: \ 0.4 \cdot 1000 = 400 & \text{Code} \\
C &: \ 0.3 \cdot 1000 = 300 & 10 \\
T &: \ 0.2 \cdot 1000 = 200 & 110 \\
A &: \ 0.1 \cdot 1000 = 100 & 111 \\
\end{align*}
\]

Number of bits = 400 \cdot 1 + 300 \cdot 2 + 200 \cdot 3 + 100 \cdot 3

\[
= 400 + 600 + 600 + 300
\]

\[
= 1900 \ \text{bits}
\]

What if we used a fixed-length code?

Four characters only needs 2-bit each

\[
\begin{array}{cccc}
G & C & T & A \\
00 & 01 & 10 & 11 \\
\end{array}
\]

1,000 characters = 2,000 bits
What if frequencies were

\[
\begin{array}{cccc}
G & C & T & A \\
0.8 & 0.1 & 0.05 & 0.05 \\
\end{array}
\]

1,000 characters = \(800 \cdot 1 + 100 \cdot 2 + 50 \cdot 3 + 50 \cdot 3\)
\[= 800 + 200 + 150 + 150 = 1,300\]

fixed length still \(2,000\)

Savings increases with number of characters.
How to construct a Huffman code?

or

How to construct that decoding binary tree?

Start with a leaf for each character with its frequency.

\[
\begin{align*}
    & \text{G} \quad \frac{0.4}{4} \\
    & \text{C} \quad \frac{0.3}{3} \\
    & \text{T} \quad \frac{0.2}{2} \\
    & \text{A} \quad \frac{0.1}{1}
\end{align*}
\]

Merge two nodes with lowest frequency. Sum their frequencies.

\[
\begin{align*}
    & \text{G} \quad \frac{0.4}{4} \\
    & \text{C} \quad \frac{0.3}{3} \\
    & \text{0.3}
\end{align*}
\]

\[
\begin{align*}
    & \text{G} \quad \frac{0.4}{4} \\
    & \text{C} \quad \frac{0.3}{3} \\
    & \text{T} \quad \frac{0.2}{2} \\
    & \text{A} \quad \frac{0.1}{1}
\end{align*}
\]

Repeat:

\[
\begin{align*}
    & \text{G} \quad \frac{0.4}{4} \\
    & \text{C} \quad \frac{0.3}{3} \\
    & \text{0.3}
\end{align*}
\]

\[
\begin{align*}
    & \text{G} \quad \frac{0.4}{4} \\
    & \text{C} \quad \frac{0.3}{3} \\
    & \text{T} \quad \frac{0.2}{2} \\
    & \text{A} \quad \frac{0.1}{1}
\end{align*}
\]
Repeat again

and assign 0 to left subtrees and 1 to right ones.
Pseudo code from page 431.

Assume Q is a min-priority queue of objects containing a frequency and, if a leaf, the character. In the queue, the order is determined by the frequency.

Let C be the initial objects containing each character and its frequency.

Each object has attributes .freq, .left, .right.

\[ n = |C| \]

\[ Q = C \]

for \( i = 1 \) to \( n-1 \)

\[ z = \text{new object} \]

\[ z.\text{left} = x = \text{pop}(Q) \]

\[ z.\text{right} = y = \text{pop}(Q) \]

\[ z.\text{freq} = x.\text{freq} + y.\text{freq} \]

\[ \text{push}(Q, z) \]

return \text{pop}(Q) \# \text{top node}
To encode a string of characters, a dictionary of Huffman codes, indexed (keyed) by characters, would be useful.

How can you construct this dictionary from the tree?
Now, how would you use this dictionary to encode a text string?

'GCCAT 6C' => 01010111110010

And how would you decode this?

01010111110010 => 'GCCAT 6C'
Why does the Huffman algorithm guarantee an optimal (minimum size) prefix-free code?

or

Does the greedy choice of merging the two lowest frequency characters always work?

**Lemma 16.2**

T is an optimal tree.

a and b are at maximum depth.

x and y are characters with lowest frequency.

Swap x and a, and y and b.

Cost of codes for $T''$ same as cost for $T$, so $T''$ is also an optimal tree.
Details: Assume $a.f\leq b.f$\\ $x.f\leq y.f$\\ 
\[B(T)\text{ is the expected cost of encodings using } T.\]
\[B(T) = \sum_{c \in C} c.f \cdot d_T(c)\]
set of all characters or number of bits in c code.

Let $T'$ be the tree after swapping just x and a.

\[B(T) - B(T') = \sum_{c \in C} c.f \cdot d_T(c) - \sum_{c \in C} c.f \cdot d_{T'}(c)\]

\[= x.f \cdot d_T(x) + a.f \cdot d_T(a) - x.f \cdot d_{T'}(x) - a.f \cdot d_{T'}(a)\]

\[= x.f \cdot d_T(x) + a.f \cdot d_T(a) - x.f \cdot d_{T'}(a) - a.f \cdot d_{T'}(x)\]

\[= (x.f - a.f) \cdot d_T(x) + (a.f - x.f) \cdot d_T(a)\]

\[= (a.f - x.f) \cdot (d_T(a) - d_T(x))\]

\[\geq 0\quad \text{nonnegative because } x \text{ is minimum frequency leaf}\]
\[\text{nonnegative because } a \text{ is maximum depth leaf in } T\]
So $B(T) - B(T') \geq 0$.

We can show in the same way that $B(T') - B(T'') \geq 0$. So $B(T) - B(T'') \geq 0$ or $B(T) \geq B(T'')$.

But, since we assumed $T$ is optimal, $B(T) \leq B(T'')$. So, $B(T) = B(T'')$!

We just showed that ...

... $T''$, formed by merging lowest frequency characters, is an optimal tree!

Of all possible mergers at each step, the Huffman algorithm chooses the one that incurs the least cost.

Now, let’s prove that the problem of constructing optimal prefix codes has the optimal substructure property.
Lemma 16.3. As before, let $x$ and $y$ be the characters with the two lowest frequencies. We must show that if $T'$ is optimal for $C - \{x, y\}$, then the step of replacing $z$ with

$$z \text{freq} = x \text{freq} + y \text{freq}$$

results in an optimal tree for $C$. Formally:

$$B(T) = \sum_{c \in C - \{x, y\}} c \text{ freq} \cdot d(c) + (x \text{ freq} + y \text{ freq}) (d(z) + 1)$$

$$\Rightarrow B(T') = B(T) - x \text{ freq} - y \text{ freq}$$
Let's assume that what we want to be true, that $T$ is optimal for $C_i$, is not true, and see if this results in a contradiction. If $T$ is not optimal for $C_i$, then for tree $T''$ that is optimal for $C_i$, $B(T'') < B(T)$.

Call $T''$ the tree that we get by merging $x$ and $y$ in $T$. $B(T'') = B(T') - x.freq - y.freq < B(T) - x.freq - y.freq = B(T')$.

So $B(T'') < B(T')$,

contradicting our assumption that $T$ and $T'$ are optimal!