Huffman Coding

Data Compression

Problem: Given a frequency distribution for alphabet characters in a particular message, determine the optimal bit-encoding of alphabet characters so as to minimize the size of the (lossless) encoded message.

Example: Assume that a message contains the following English-language character frequencies: A: 30, B: 20, C: 15, D: 10, E: 9, F: 8, G: 7, H: 1

How many bits does it take to distinguish 8 distinct characters? 3

Encoding: A: 000, B: 001, C: 010, D: 011, E: 100, F: 101, G: 110, H: 111

How many total bits will be in this message?
(100 characters)(3 bits/character)=300 bits

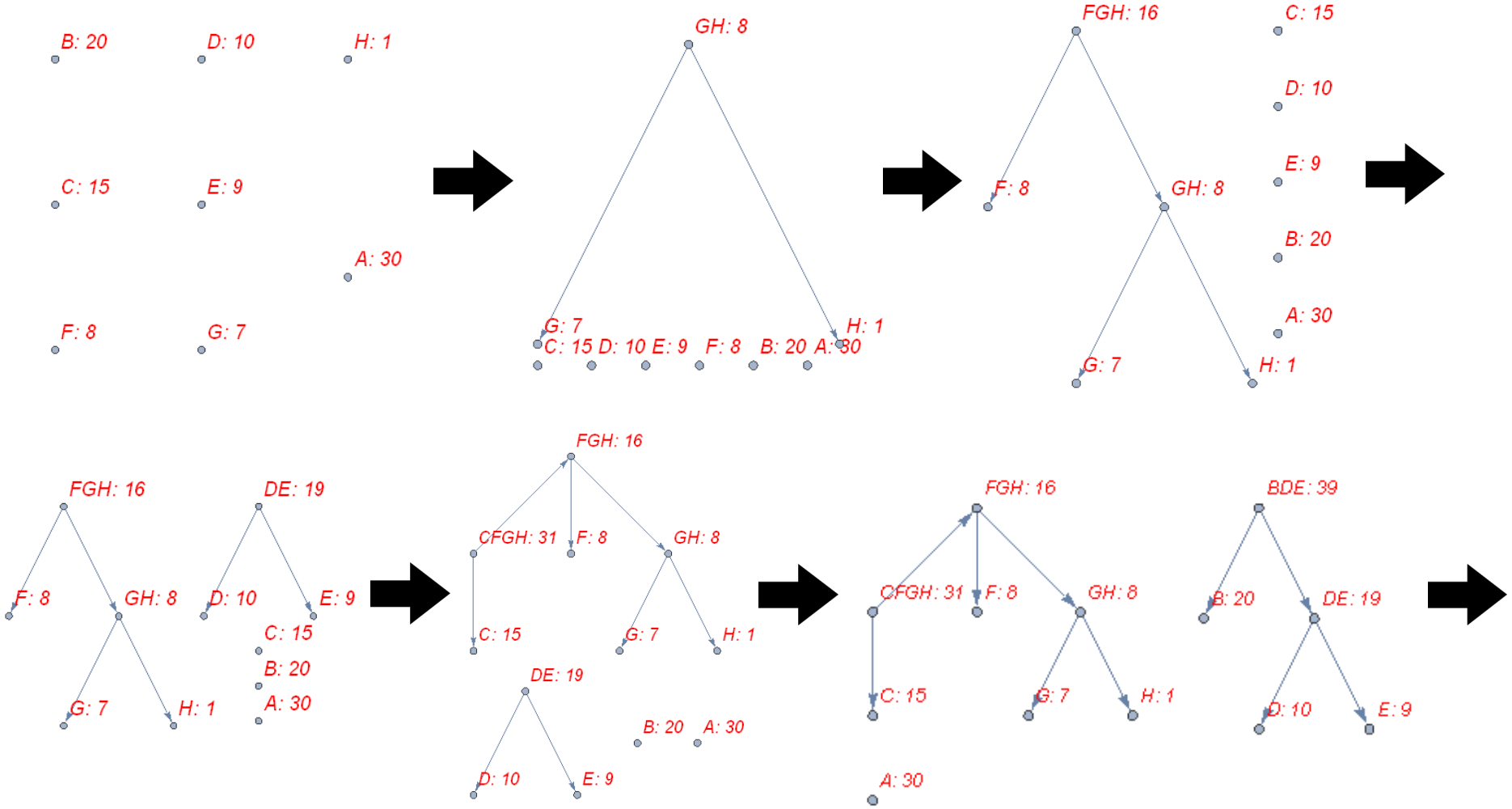
Can we do better? Consider the following bit-encoding for the characters.
Another encoding: A: 10, B: 01, C: 110, D: 001, E: 000, F: 1111, G: 11101, H: 11100

How many total bits will be in the message now?
$2(30) + 2(20) + 3(15) + 3(10) + 3(9) + 4(8) + 5(7) + 5(1) = 274$ bits

Huffman's Algorithm

Algorithm: Take the two smallest root vertices and merge them together until there is a single root.

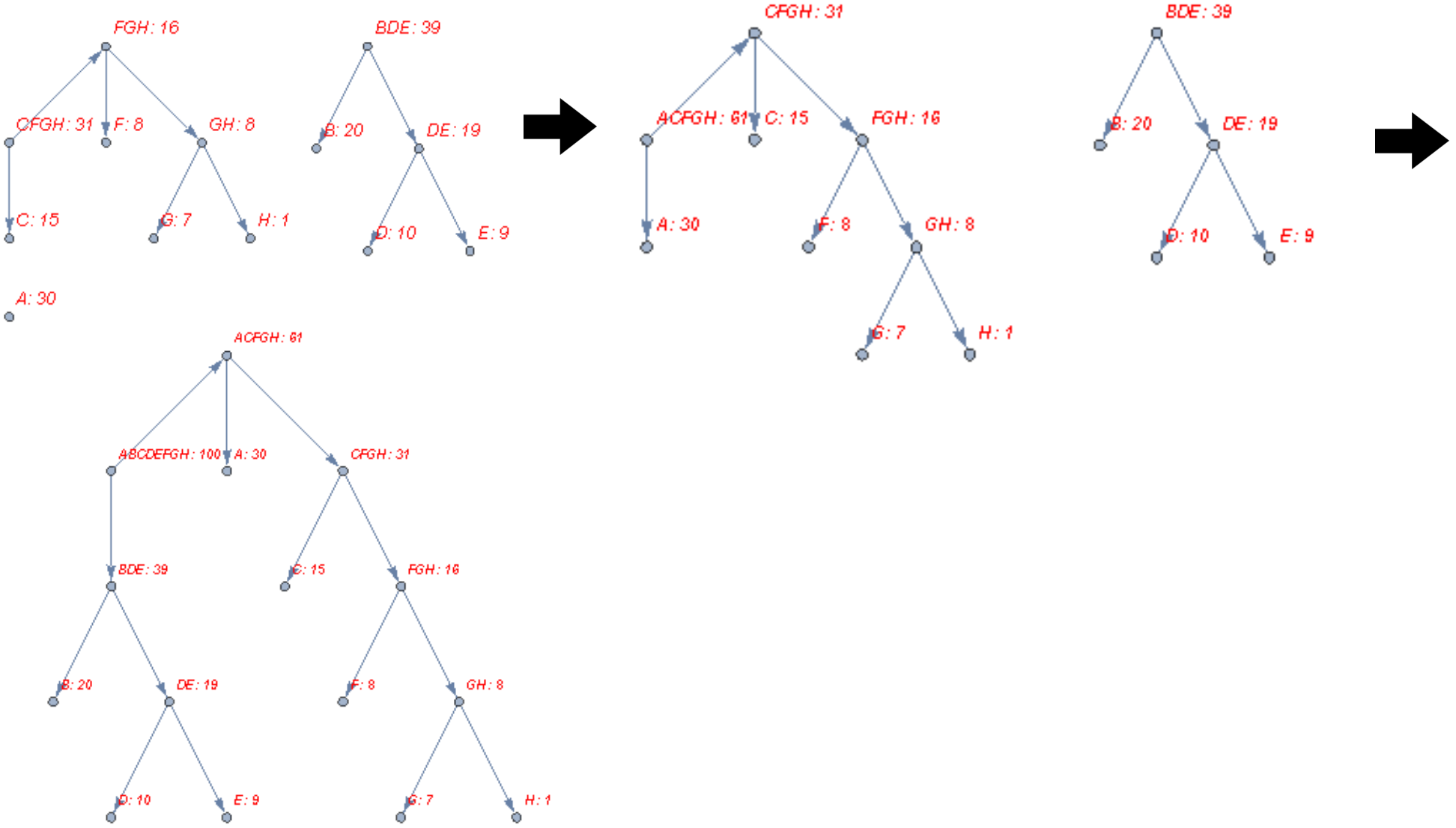
Example: A: 30, B: 20, C: 15, D: 10, E: 9, F: 8, G: 7, H: 1



Huffman's Algorithm: Continued

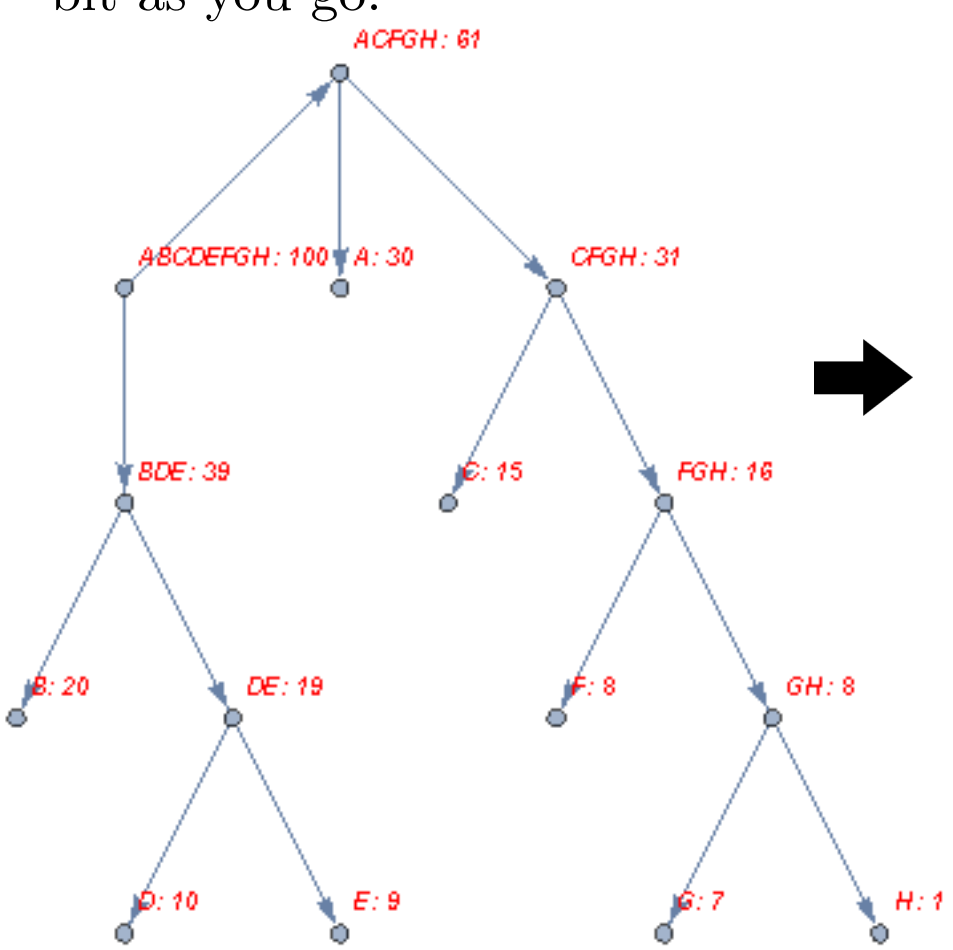
Algorithm: Take the two smallest vertices and merge them together until there is a single root.

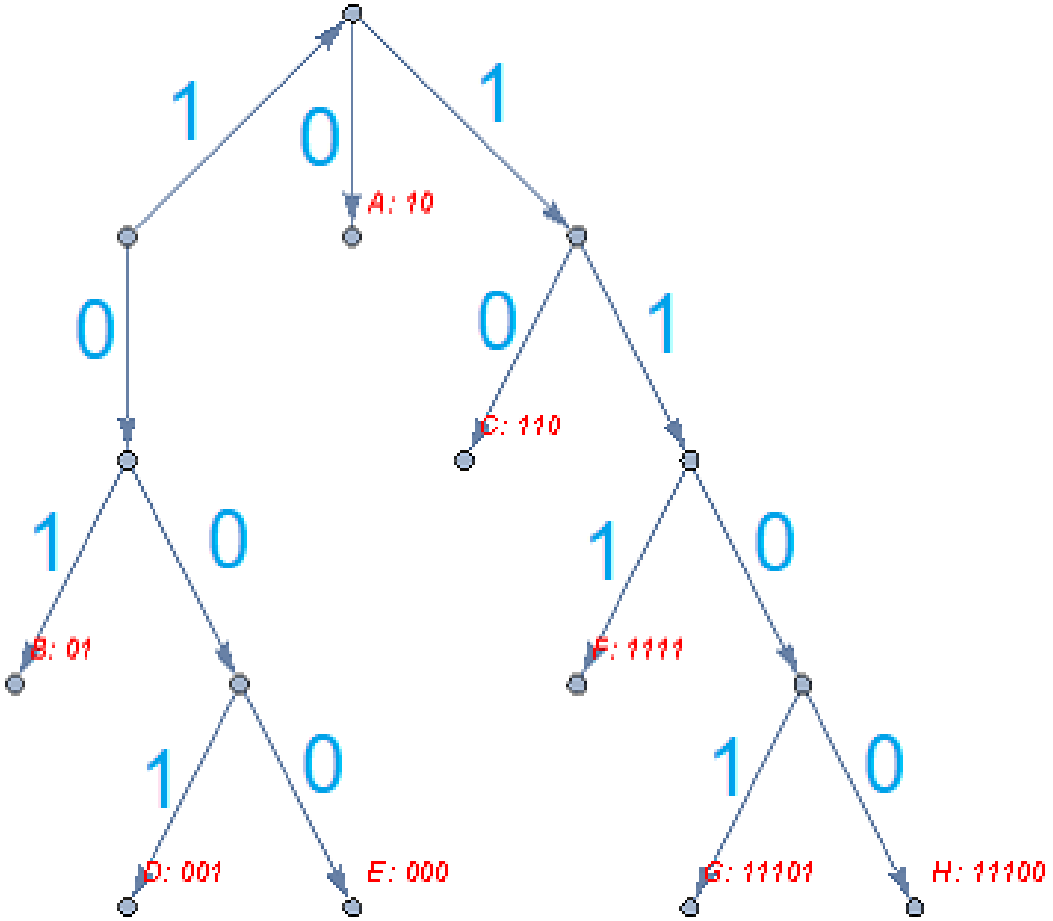
Example: A: 30, B: 20, C: 15, D: 10, E: 9, F: 8, G: 7, H: 1



Huffman Codewords

For every node, arbitrarily assign 0 and 1 to both directions leading out from the node. To assign codewords to characters, starting from the root (ABCDEFGH), follow the path down to the appropriate leaf/character, adding the appropriate bit as you go.





Encoding and Decoding

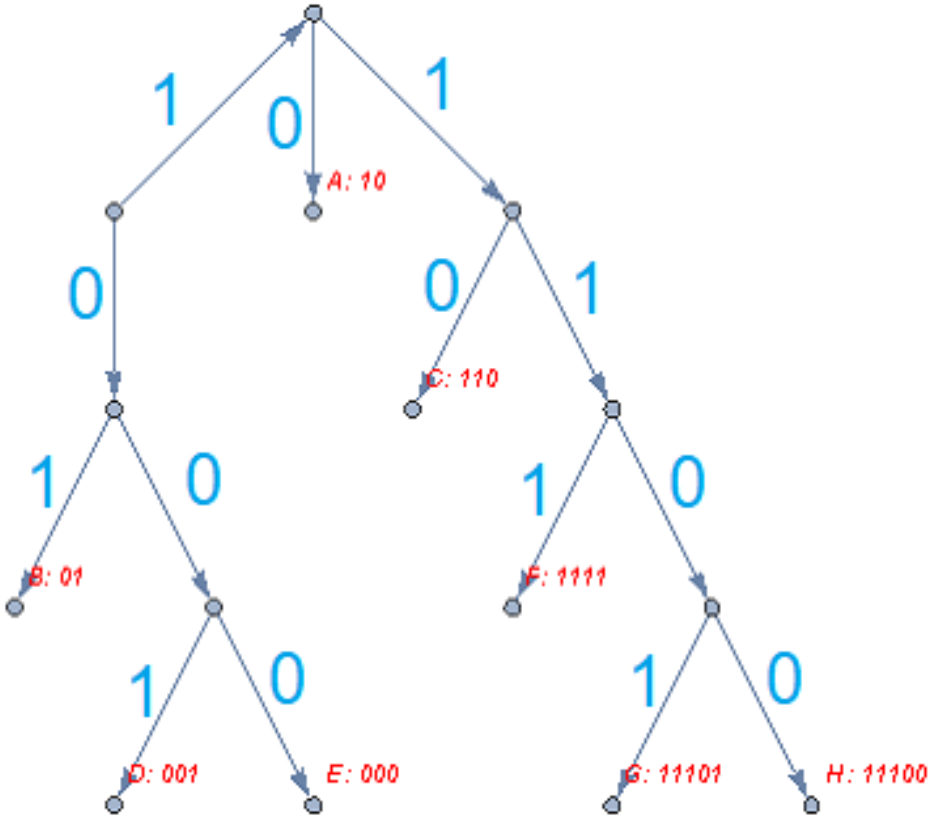
A: 10, B: 01, C: 110, D: 001, E: 000, F: 1111, G: 11101, H: 11100

Using the Huffman code that we generated, we can encode messages.

CDADEBADGH: 1100011000100001100011110111100

Given a message, how do we decode? Start at the root of the tree and follow the path to the appropriate character.

$110|001|10|001|000|01|10|001|11101|11100 = C|D|A|D|E|B|A|D|G|H$



A prefix code is defined to be a code where no codeword is the prefix of any other.

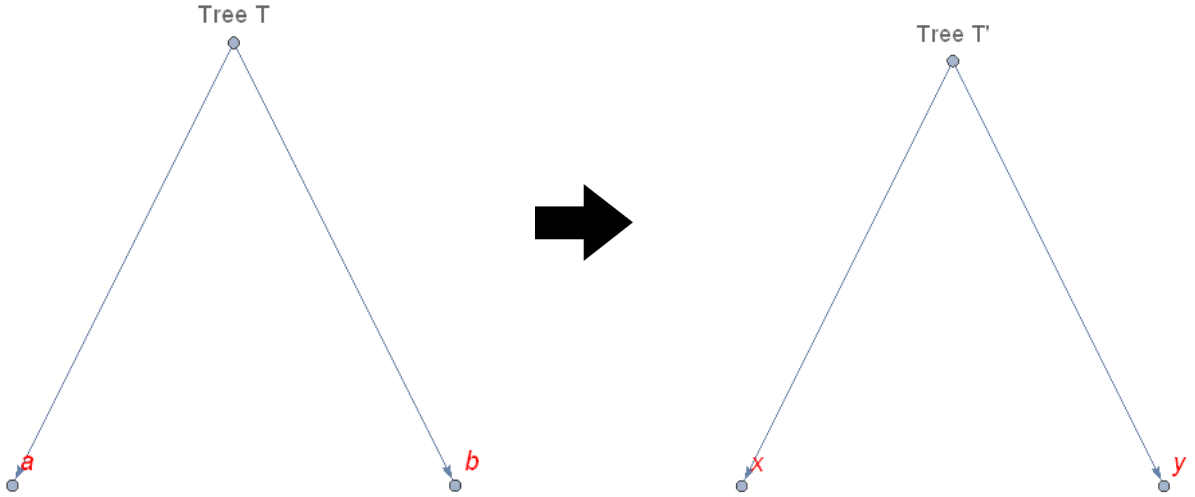
Claim: *Every* Huffman code is a prefix code.

Optimality

Lemma: Let x and y be the two characters with lowest frequency in the alphabet. Then there exists an optimal prefix code in which the codewords for x and y are siblings in the deepest leaves of the code tree.

Proof: Let a and b be two sibling characters of T, the optimal tree, that appear in the deepest leaves of the code tree. (Note that a and b are guaranteed to exist because there will never be a leaf of the tree without a sibling: it would violate the optimality of the tree T.)

WLOG, assume that $f_a \leq f_b$ and $f_x \leq f_y$. Because x and y are the two lowest frequency characters, we must have $f_x \leq f_a$ and $f_y \leq f_b$. Exchange the positions of $a \leftrightarrow x$ and $b \leftrightarrow y$ to get a new tree T'.



Optimality

Now we want to compare the number of bits in a given message if we use T' as opposed to T.

$$\text{total bits using } T - \text{total bits using } T' =$$

$$\sum_{t \in \text{leaves of } T} f_t \cdot d_T(t) - \sum_{t \in \text{leaves of } T'} f_t \cdot d_{T'}(t) =$$

$$= f_x \cdot d_T(x) + f_a \cdot d_T(a) - f_x \cdot d_{T'}(x) - f_a \cdot d_{T'}(a)$$

$$+ f_y \cdot d_T(y) + f_b \cdot d_T(b) - f_y \cdot d_{T'}(y) - f_b \cdot d_{T'}(b)$$

$$= f_x \cdot d_T(x) + f_a \cdot d_T(a) - f_x \cdot d_T(a) - f_a \cdot d_T(x)$$

$$+ f_y \cdot d_T(y) + f_b \cdot d_T(b) - f_y \cdot d_T(b) - f_b \cdot d_T(y)$$

$$= (f_a - f_x)(d_T(a) - d_T(x)) + (f_b - f_y)(d_T(b) - d_T(y)) \geq 0$$

Which tree seems to be more efficient? T'

T is an optimal tree$\Rightarrow$ T' must also be optimal.

Optimality

Lemma: Assume that we begin with an alphabet A that contains two minimal-frequency characters x and y. If we are given an optimal tree T' for the alphabet A', where the two characters x and y are merged into $\{x, y\}$, then the tree T formed by replacing the node $\{x, y\}$ with an internal node with two children, x and y, is also optimal for the original alphabet A.

Question: How many additional bits do we need to encode a message using alphabet A with tree T as opposed to alphabet A' with tree T'? (Note that less information will be sent with alphabet A' (Why?) so we expect fewer bits.)

$$\text{total bits using } T - \text{total bits using } T' =$$

$$\sum_{t \in \text{leaves of } T} f_t \cdot d_T(t) - \sum_{t \in \text{leaves of } T'} f_t \cdot d_{T'}(t) =$$

$$= f_x d_T(x) + f_y d_T(y) - (f_x + f_y) d_{T'}(x + y)$$

$$= (f_x + f_y) d_T(x) - (f_x + f_y)(d_T(x) - 1) = f_x + f_y$$

Optimality

Consider any optimal tree S for A. We know by the first Lemma that x and y can be assumed to be sibling leaves. Let S' be the tree that you get by merging x and y. Then we get

$$\begin{aligned}
\sum_{t \in \text{leaves of } S} f_t \cdot d_S(t) &= \sum_{t \in \text{leaves of } S'} f_t \cdot d_{S'}(t) + f_x + f_y \\
&\geq \sum_{t \in \text{leaves of } T'} f_t \cdot d_{T'}(t) + f_x + f_y \\
&= \sum_{t \in \text{leaves of } T} f_t \cdot d_T(t)
\end{aligned}$$

Because we assumed that S is optimal, T must be as well.

Optimality

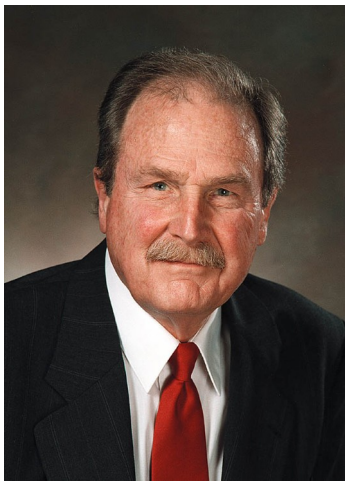
Huffman's Theorem: Huffman's code always produces the optimal code (among binary codes that encode individual characters independently).
Proof: Mathematical induction on the size of the alphabet. If the alphabet has size 2, then Huffman's code is trivially optimal.
Assume that Huffman's algorithm produces an optimal tree if the size of the alphabet is k.
Assume that $|A| = k + 1$. Merge the two lowest frequency characters into one to get A'. Note that Huffman's algorithm operates by merging the two lowest frequency characters first; Huffman would have proceeded from A to A'.
$|A'| = k \rightarrow$Huffman's algorithm produces an optimal tree for A'.
By the Lemma, expanding the tree yields an optimal tree for A.

History

In 1951, Huffman (MIT, information theory class) was given the choice of a term paper or a final exam. The professor, Robert Fano, assigned a term paper on the problem of finding the most efficient binary code.
Robert Fano had worked with information theory pioneer Claude Shannon to develop a similar code. Huffman avoided the major flaw of the suboptimal Shannon-Fano coding by building the tree from the bottom up instead of from the top down.

Huffman never patented his idea.

Applications: communications protocols, GIF (visual), MP3 (audio), MP4 (movies), ZIP (Microsoft), Brotli (Google), etc.



David Huffman

Robert Fano