

File Compression using Huffman Coding

A Huffman Code is an optimal prefix code, that guarantees unique decodability of a file compressed using the code. The code was devised by Huffman as part of a course assignment at MIT in the early 1950s.

Huffman coding is a technique for assigning binary sequences to elements of an alphabet. The goal of an optimal code is to assign the minimum number of bits to each symbol (letter) in the alphabet.

ASCII is an example of a *fixed length code*. There are 100 printable characters in the ASCII character set, and a few non printable characters, giving 128 total characters. Since $\lg 128 = 7$, ASCII requires 7 bits to represent each character. The ASCII character set treats each character in the alphabet equally, and makes no assumptions about the frequency with which each character occurs.

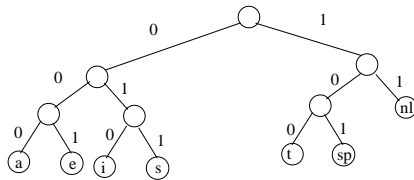
A *variable length code* is based on the idea that for a given alphabet, some letters occur more frequently than others. This is the basis for much of information theory, and this fact is exploited in compression algorithms to use as few bits as possible to encode data without "losing" information. More sophisticated compression techniques can use compression techniques that actually discard information. For example, image and video data can take a sustain a certain amount of loss since our brain can compensate for missing information, up to a degree.

However, for text compression, we don't want to have characters discarded as part of the compression, so a text compression requires a unique decodability condition of the compression algorithm.

The Huffman coding algorithm produces a optimal variable length prefix code for a given alphabet in which frequencies are preassigned to each letter in the alphabet. Symbols that occur more frequently have a shorter codewords than symbols that occur less frequently. The two symbols that occur least frequently will have the same codeword length.

Example

We can represent the code using a binary tree where a left branch is labeled with a 0 and a right branch is labeled with a 1. Note that all the symbols are at the leaf nodes and that this binary tree is a full tree---all nodes are either leaves, or have two children.



As long as no character code is a prefix of another code, we can define a more optimal variable length code that is called a prefix code. A prefix code has to assign to each letter a unique prefix so that given a codeword, the letter can be uniquely determined by traversing the tree using each bit of the codeword to determine which branch to follow.

Given that we know the frequencies of the symbols in the alphabet, we can use Huffman's algorithm to find a prefix code using an optimal number of bits.

Example

Suppose we have a simple alphabet with the following n characters and the assigned frequency counts, and we use a fixed length code requiring $\lceil \lg n \rceil$ bits for each character.

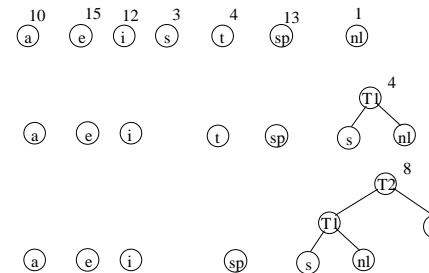
Fixed Length Coding Scheme Ignoring Frequency

Character	Code	Freq	Bits
a	000	10	30
e	001	15	45
i	010	12	36
s	011	3	9
t	100	4	12
sp	101	13	39
nl	110	1	3
Total			174

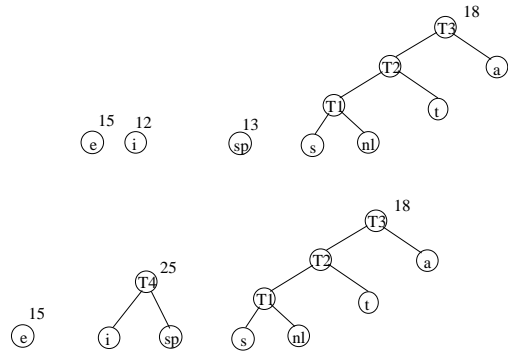
Huffman's Algorithm

The algorithm works by constructing a binary tree from the bottom up, using the frequency counts of the symbols to repeatedly merge subtrees together. Intuitively, the symbols that are more frequent should occur higher in the tree and the symbols that are less frequent should be lower in the tree.

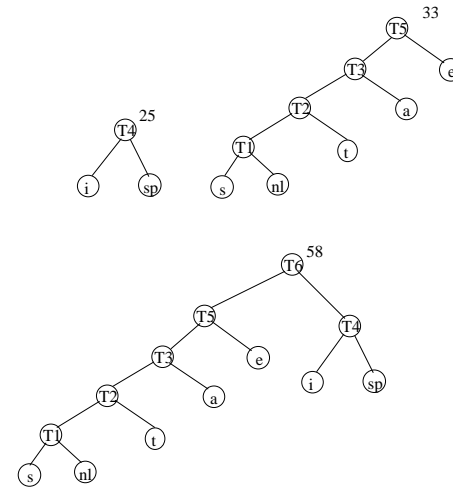
Conceptually, the algorithm creates a weighted node for each symbol, and repeatedly merges the lowest frequency nodes into the tree, adding the weights cumulatively.



Huffman's Algorithm



Huffman's Algorithm.



Huffman's Algorithm.

Optimal Variable Length Code

Character	Code	Freq	Bits
a	001	10	30
e	01	15	30
i	10	12	24
s	00000	3	15
t	0001	4	16
sp	11	13	26
nl	00001	1	5
Total			146

Huffman's Algorithm

Assigning frequency counts to symbols in an alphabet requires probability analysis for a given alphabet. It is also the case that a compression algorithm could scan a given file and compute the frequencies dynamically. However, this may require significant computation since a file may be very large (several megabytes). A good compression program can instead repeatedly compute frequencies on fixed size blocks of texts and store the code as part of the compressed file.

Once the Huffman codes are calculated, the tree structure can be represented using an array of 255 bytes, where bytes 0-127. Within each byte, bit 8 is either 0 or 1 indicating that the character is the left or right child of a parent node, whose index is represented in the next 7 bits. The index value + 128 gives the position in the upper half of the array (128-255) where the parent node is stored. The high order bit of the parent entry is again a 0 or 1 and the lower 7 bits is the index for its parent. The root node has a parent index of 0.

This way, the entire code can be represented in 255 bytes, which is a small overhead that can be inserted into the front of the compressed file as part of encoding.

On decoding, the uncompress program uses the table to determine an entry number 0-127 for each prefix code, and produces that ASCII character as output.