

Huffman Encoding

The data transformation process for PocketBrief uses the *Huffman Encoding Algorithm* for compression and decompression of data, respectively. The details of the algorithm are given below.

Huffman encoding is a simple compression algorithm introduced by **David Huffman** in 1952 and makes use of a binary tree to develop codes of varying lengths for the letters used in the original message.

Working of Huffman Encoding

A sample string, “**traversing threaded binary trees**”, is used as an example for demonstrating working of *Huffman Encoding*. The given string is of length 32 characters.

Supposing the phrase is to be sent using standard 8-bit ASCII codes, it would be required to send $8 \times 32 = 256$ bits. Given that there are only fourteen different characters in the phrase, each could be represented with just four bits per character (since $2^4 = 16$, which is larger than 14). Then only 128 bits would require to be sent. Using the variable length codes in the *Huffman algorithm*, we can reduce the size by an additional ten percent, sending only 116 bits.

To form these codes, a binary tree structure is needed. This binary tree differs from standard binary trees by the fact that it is most easily constructed from the bottom to the top: from the leaves to the root, in other words. The procedure is as follows: first, list all the letters used, including the "space" character, along with the frequency with which they occur in the message. Consider each of these character/frequency pairs to be nodes, which are actually leaf nodes. Pick the two nodes with the lowest frequency, and if there is a tie, pick randomly amongst those with equal frequencies. Make a new node out of these two, and make the two nodes its children. This new node is assigned the sum of the frequencies of its children. Continue the process of combining the two nodes of lowest frequency until only one node, the root, remains.

Encoding is simple: for each character it is necessary to send the codes as determined above. For example, to send "tra", the first three letters in the message, the code 1110100010 would be sent. It may seem, at first glance, to be impossible to decode such a message since each letter is of varying length. This is not at all the case. The recipient of the message simply starts at the root of the tree and uses the numbers to arrive at a leaf node. At that point the letter in the leaf node is recorded, and the receiver starts again at the root. In the given example, the receiver would follow three branches to the right (because of the three 1's) before arriving at the letter 't'. Returning to the root, the receiver would follow one branch left, then one right, then another left before arriving at the 'r'. And so forth.

Space saving effected by *Huffman encoding* varies depending upon the data, anywhere from 70% over ASCII text and 25% over fixed-length codes.

Huffman encoding is not, of course, the "perfect" solution. In order for the coded text to be successfully understood, a header containing the list of codes (a= 0010, b= 10100, etc) must be sent with each file, thus slightly reducing the savings for smaller files. This can, of course, be avoided by using a standard set of codes based upon average frequencies in the English language.

The other major drawback of *Huffman encoding* is the possibility of data corruption. If a single bit is switched when the data is transmitted, the message will likely be entirely corrupted. In our example, if the first bit was switched from a 1 to a 0, the message would begin "yenseb..." and continue unintelligibly. The possibility also exists that the message will just be partially corrupted. Suppose the third bit was switched from a 1 to a 0. Then the message would begin " raversing..." and continue correctly, the only difference being that a space was substituted as the first character. Nevertheless, if a bit is switched, the possibility for major corruption is most probable.

The *Huffman* algorithm is used in a number of commercial compression programs and is also a part of the JPEG image compression algorithm.