http://www.programminglogic.com/implementing-huffman-coding-in-c/

[**Programming Logic**](http://www.programminglogic.com/)

**Algorithms, Computer Science and Programming Puzzles**

**Implementing Huffman Coding in C**

[Huffman Coding](http://en.wikipedia.org/wiki/Huffman_coding) (link to Wikipedia) is a compression algorithm used for loss-less data compression. Here’s the basic idea: each ASCII character is usually represented with 8 bits, but if we had a text filed composed of only the lowercase a-z letters we could represent each character with only 5 bits (i.e., 2^5 = 32, which is enough to represent 26 values), thus reducing the overall memory needed to store the data.

For example, the table of characters -> binary code could look like this:

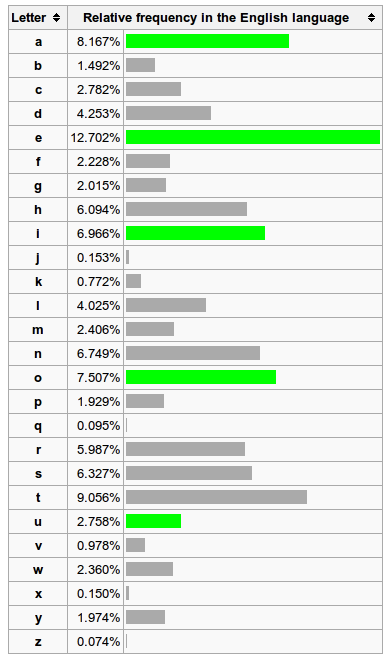
* a = 00000
* b = 00001
* c = 00010
* d = 00011
* e = 00100

And so on.

This is the fixed-length representation, and it already creates a significant compression rate (around 35% in the example above).

A more efficient approach is to use a variable-length representation, where each character can have a different number of bits to represent it. More specifically we first analyze the frequency of each character in the text, and then we create a binary tree (called Huffman tree) giving a shorter bit representation to the most used characters, so that they can be reached faster. Notice that it must be a prefix tree (i.e., the code of every letter can’t be prefix to the code of any other letter) else the decompression wouldn’t work.

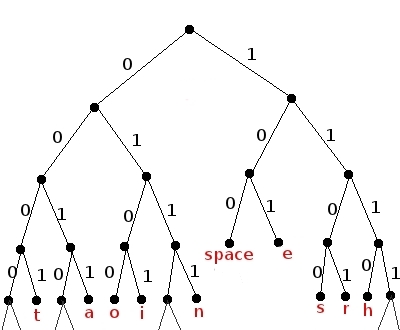
For example, the frequency of the letters in the English language (according to Wikipedia) is the following:



Now the algorithm to create the Huffman tree is the following:

* Create a forest with one tree for each letter and its respective frequency as value
* Join the two trees with the lowest value, removing each from the forest and adding instead the resulting combined tree
* Repeat until there’s only one tree left

The Huffman tree for the a-z letters (and the space character) using the frequency table above would look like this (you would need to expand to the lower branches to see all the letters):



We can traverse the tree and create a table with all the letters and their respective binary codes. It would look like this:

a = 0011

b = 011011

c = 11111

d = 00100

e = 101

f = 000010

g = 011001

h = 1110

i = 0101

j = 011010101

k = 0110101001

l = 00101

m = 000001

n = 0111

o = 0100

p = 011000

q = 0110101000

r = 1101

s = 1100

t = 0001

u = 11110

v = 0110100

w = 000000

x = 011010111

y = 000011

z = 011010110

space = 100

After that you just need to read the original text file letter by letter and to output the respective binary code. If you want to decompress a file, on the other hand, you read bit by bit and move along the Huffman tree until you find a letter, at which point you move back to the root of the tree and continue processing the bits.

Below you’ll find a C implementing of the Huffman coding (it includes all the parts, including the creating of the Huffman tree, the table and so on). If you prefer here’s the [huffman.zip](http://www.programminglogic.com/downloads/huffman.zip) file for download.