Text Compression by Static Huffman Coding

Jeny Rajan
Department of Computer Science, University of Kerala

Basically data compression algorithms can be of two types (i) Lossless data compression and (ii) Lossy data compression. Huffman coding algorithm is one of the oldest and popular lossless data compression algorithm. The Huffman coding was developed by David Huffman as part of a class assignment taught by Robert Fano at MIT in 1951. Probably it may be the best-known coding method based on probability statistics.

Huffman coding assigns an output code to each symbol, with the output codes being as short as 1 bit, or considerably longer than the input symbols, strictly depending on their probabilities. The optimal number of bits to be used for each symbol is the log base 2 of (1/p), where p is the probability of a given character. Thus, if the probability of a character is 1/256, such as would be found in a random byte stream, the optimal number of bits per character is log base 2 of 256, or 8. If the probability goes up to 1/2, the optimum number of bits needed to code the character would go down to 1[3]. In ASCII coding, we are using 8 bits per character. Huffman coding compress data by using fewer bits to encode motre frequently occurring characters so that not all character are encoded with 8 bits [4].

Entropy

Shannon borrowed the definition of entropy from statistical physics to capture the notion of how much information is contained in a source and their probabilities. For a set of messages S, Shannon defined entropy as,

$$H(S) = \sum_{s \in S} p(s) \log_2 \frac{1}{p(s)}$$

where P(s) is the probability of the message s.
If we consider the individual messages s∈ S, Shannon defined the notion of the self information of a message as

$$i(s) = \log_2 \frac{1}{p(s)}$$

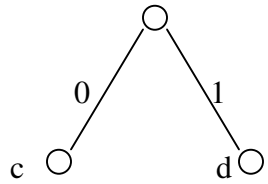
This self information representation the number of bits of information contained in it and, roughly speaking, the number of bits we should use to send that message. The entropy is simply a weighted average of the information of each message, and therefore the average number of bits of information in the set of messages[5]. It is possible to compress the source, in a lossless manner, with compression rate close to H. It is mathematically impossible to do better than H[6]. The ratio of the entropy of a source to the maximum value it could have while still restricted to the same symbols will be called its relative entropy. This is the maximum compression possible when we encode into the same alphabet. One minus the relative entropy is the redundancy. The redundancy of ordinary English, not considering statistical structure over greater distances than about eight letters, is roughly 50% [2].

Constructing Huffman Tree

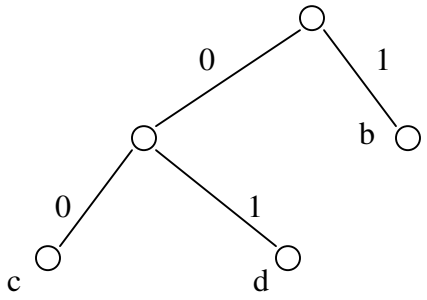
Let the message to be coded is “aaabbcd”. Then the probability of the characters will be as follows

P(a)=0.33
P(b)=0.22
P(c)=0.11
P(d)=0.11

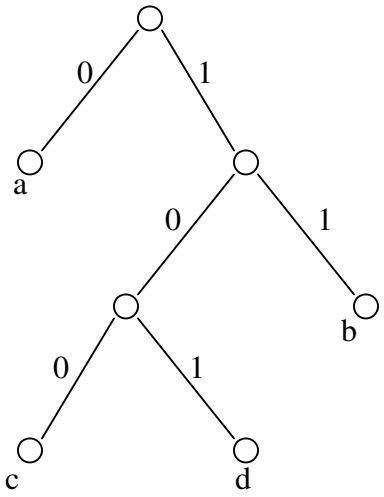
Here the characters with lowest probabilities are “c” and “d”. So we can put “c” and “d” as the leaves of the Huffman tree as below. Put 0 to the left node and 1 to the right node. Probabilities of left node shall be less than or equal to the probabilities of the right node. So after first step the tree will be as follows



The resultant probability will be p(a)+p(b)=0.22. Next we have to take the lowest from 0.22,0.22 and 0.33.So we have to take the first 2. The result will be as follows.



Now the resultant probability is 0.44 which is higher then 0.33, so the resultant tree will be



So the Huffman code for a,b,c,d will be
a = 0
b = 1 1
c = 1 0 0
d= 1 0 1
Now the Huffman code for for the string "aaabbcd" will be
0001111100101 – total 13 bits
The ASCII code for the string "aaabbcd" is
and consists of 7 x 8 =56 bits. So the compression ratio will be (1- (13/56)) =0.7679

ie., the resultant file is only 20% of the original file.

Huffman Coding Algorithm

1. Arrange the symbol probabilities P_i in a decreasing order and consider them as the leaf nodes of a tree.
2. While there is more than one node :
    - Merge the two nodes with smallest probability to form a new node whose probability is the sum of the two merged nodes.
    - Arbitarily assign 1 and 0 to each pair of branches merging into a node.

3. Read sequentially from the root node to the leaf node where the symbol is located.

The preceding algorithm gives the Huffman codebook for any given set of probabilities. Coding and decoding is done simply by looking up values in a table [8]. For a code of size *n* this algorithm will require n-1 steps since every complete binary tree with *n* leaves has *n-1* internal nodes, and each step creates one internal node. If we use a priority queue with *O(log n)* time insertions and find-mins(eg, a heap) the algorithm will run in *O(n log n)* time [5]. The key property of Huffman codes is that they generate optimal prefix codes. We show this in the following theorem, originally given by Huffman.

The Huffman algorithm generates an optimal prefix code.

The following restrictions will be imposed on Huffman code for generating optimal prefix code[1]

a) No two messages will consist of identical arrangements of coding digits
b) The message codes will be constructed in such a way that no additional indication is necessary to specify where a message code begins and ends once the starting point of a sequence of messages is known.
c) L(1)<=L(2)<=…….<=L(N-1)=L(N), where L(i) is the length of the message.
d) At least two or not more than D of the messages with code length L(N) have codes which are alike except for their final digits.
e) Each possible sequence of L(N)-1 digits must be used either as a message code or must have one of its prefixes used as a message code.

Experimental Results

Source File	Original Size	After Compression	Compression Ratio
Readme.txt	2527 bytes	2173 bytes	0.1409
Paper4.txt	13286 bytes	8709 bytes	0.3445
Paper5.txt	12288 bytes	8396 bytes	0.3168
Table.txt	1194 bytes	629 bytes	0.4732
Tx1.txt	2789 bytes	1365 bytes	0.5106
Average Compression			**0.30378**

The time complexity of the Huffman algorithm is **O(nlogn)**. Using a heap to store the weight of each tree, each iteration requires **O(logn)** time to determine the cheapest weight and insert the new weight. There are **O(n)** iterations, one for each item.

Conclusion

Our experimental results show that with Huffman coding about 30% compression can be achieved. The problem with Huffman coding lies in the fact that Huffman codes have to be an integral number of bits long. For example, if the probability of a character is 1/3, the optimum number of bits to code that character is around 1.6. The Huffman coding scheme has to assign either 1 or 2 bits to code, and either leads to a longer compressed message than is theoretically possible. The non-optimal coding becomes a noticeable problem when the probability of a character becomes very high. If a stastical method can be developed that can assign a 90% probability to a given character, the optimal code size would be 0.15 bits. The Huffman coding system would probably assign a one-bit code to the symbol, which is 6 times longer than is necessary [3].

References

[1] David A. Huffman (1952), *A Method for the Construction of Minimum-Redundancy Codes,* Proceedings of the I.R.E

[2] C.E Shannon (1948), *A Mathematical Theory of Communication*, The Bell System Technical Journal, Vol. 27, pp. 379-423, 623-656.

[3] Mark Nelson (1991), *Arithmetic Coding + Statistical Modeling = Data Compression*, Dr. Dobb's Journal.

[4] Owen L. Astrachan (2001), *Huffman Coding : A CS2 Assignment*, http://www.cs.duke.edu /csed /poop/huff/info/

[5] Guy E Belelloch (2001), *Introduction to Data Compression*, www-2.cs.cmu.edu /afs/cs.cmu.edu /project/pscico-guyb/ realworld/www/compression.pdf –

[6] Nam Phamdo, *Theory of Data Compression*, http://www.data-compression.com/theory.html

[7] Nam Phamdo, *Lossless Data Compression*, http://www.data-compression.com/lossless.html

[8] Anil K. Jain, *Fundamentals of Digital Image Processing*, Second Edition, PHI