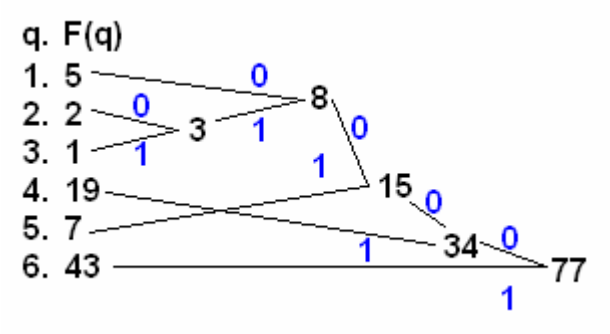
Xing Yuan
Spring 2007
18.304
Professor Kleitman

Compression Algorithms

There are many applications to being able to compress patterns or messages into shorter segments without destroying certain information about the message. For example, one may want to find a more concise method of expressing the number of times x different words is used in a paper. Amongst the plethora of compression algorithms, two algorithms stand out as being the most efficient of processes. The first of such algorithms, the simpler version, is called the Huffman algorithm; and the second algorithm is known as the Hu-Tucker algorithms.

The Huffman algorithm is based on the concept that within any subject of compression, the most efficient method is to assign the shortest code words to the most frequently appearing segments of the message. Relating to the example above for compressing the way to express the number of times different words appear in a paper, the most frequently used words would have the shortest code words such that the total length of the code would be minimized. The way in which the Huffman algorithm accomplishes this task is with the structure of a binary tree. Let us assume that the message we wish to encode consists of six words, each of which has a different frequency of usage within the paper. By arranging the words and their frequencies in order, we can form a binary tree by branching together the lowest frequency nodes and repeating the branching process for the next lowest node or combined node. See below for an example:



By completing the binary tree, we can proceed with the coding algorithms by labeling the series of branches in the tree. Assume that the upper branches are labeled 0's and the lower branches are labeled 1's. With this, the last step in the coding procedure is to start at the final node and trace each path back to the frequencies of each word in the original message. For example, the first word listed will be assigned the code 0000, and words 2 through 6 will be coded 0010, 00011, 01, 001, and 1, respectively. As a sanity check, the propriety of the Huffman codes can be verified by seeing that the word with the highest frequency, word number 6, was given a codeword of 1, and the least frequent word, word 3, has a relatively longer code of 00011. As a result of this algorithm's branching structure, each outcome will guarantee to have two words having equal lengths of the longest code words. By following this algorithm, the original message may be fully compressed as efficiently as possible.

Having seen what the Huffman algorithm can do, it is now appropriate to discuss the importance of another algorithm called the Hu-Tucker algorithm. As a variation of the Huffman algorithm, the Hu-Tucker algorithm not only compresses a message, but it also preserves the order of the codes such that the coding algorithm will ensure consistency and repetition for the code words. Rather than always merging the lowest frequency nodes, the Hu-Tucker algorithm uses a compatibility rule for the merging of

nodes. The rule claims that one should only merge nodes if there are no original nodes between them and that merges should only occur when nodes are mutually compatible with each other. Upon completing the binary tree, label the total length of the path between each word and the final node. The subsequent coding process must also abide to a set of rules:

1 Code the first word with all 0's matching the length of its path to the final node.

2 Throw away any 1's at the very end of the previous codeword.

3 Convert the last 0 to a 1.

4 Add additional 0's to the end of the codeword if necessary to make the length of the word equal to the length of the block.

The resulting outcome not only efficiently compresses the information of the message, it preserves the order in which the words were arranged in the original state.

References

1. Lii, Jonathan. “Finding Efficient Compressions; Huffman and Hu-Tucker Algorithms.” 18.310 Lectures Notes. 2006. Massachusetts Institute of Technology. 20 Mar. 2007 http://ocw.mit.edu/NR/rdonlyres/Mathematics/18-310Fall-2004/2DAD8E83-D4DE-4938-92F9-6D77EE72E7D4/0/fidng_effic_comp.pdf