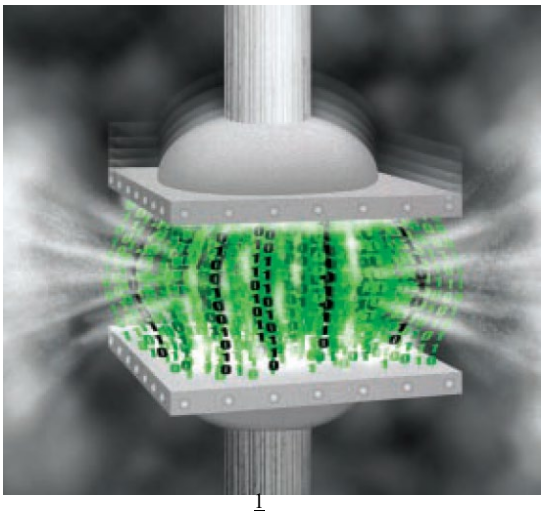
COMP 2121
DISCRETE MATHEMATICS

Lecture 16


[1]

Weighted Trees and Prefix Codes

To represent symbols, computers use strings of 0s and 1s called **codewords**. For example, in the ASCII (American Standard Code for Information Interexchange) code, the letter A is represented by the codeword 01000001, B by 01000010 and C by 01000011. In this system each symbol is represented by some string of eight bits. Therefore, string 010000110100000101000010 is decoded as CAB.

For many purposes, this kind of representation works well. However, there are situations, as in large volume storage, where this is not an efficient method. In a fixed length representation, such as ASCII, every symbol is represented by a codeword of the same length. A more efficient approach is to use codewords of variable lengths, where the symbols used most often have shorter codewords than symbols used less frequently. For example, in normal English usage, letters E, T, O and A are used much more frequently than letters Q, J, X and Z[2].

Suppose that the messages use only eight characters: E, T, O, A, Q, J, X and Z. A natural assignment to try is

E	0
T	1
O	01
A	11
Q	00
J	10
X	101
Z	011

[1] http://www.ams.org/mathmoments/mm41-compression.pdf
[2] List of letters sorted with respect to their relative frequency in the English dictionary is:
E T A O I N S R H L D C U M F P G W Y B V K X J Q Z.
List of letters sorted with respect to their relative frequency in the General Fiction (English) is:
E T A O H N I S R D L U W M C G F Y P V K B J X Z Q .

Then message ZETA can be encoded as:

0110111

However, how should the string 0110111 be decoded? Should we start looking at only the first digit, or the first two, or the first three? Depending upon the number of the first digits used, the first letter could be E, O or Z. Therefore, in order to use variable length codewords, we need to select representations that permit unambiguous decoding.

A way to do this is to construct codewords so that **no codeword is the first part of any other codeword**. Such a set of codewords is said to have the **prefix property**.

Example 1. Consider the set of codewords, S = {000, 001, 01, 10, 11}. Does S have a prefix property?

The method for decoding a string of 0s and 1s into codewords that have the prefix property is to read one digit at a time until this string of digits becomes a codeword, then repeat the process, starting with the next digit, and continue until the decoding is done.

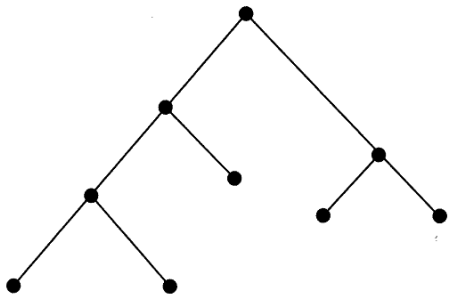
Example 2. Using the set of codewords from the Example 1, as B = 01, I = 10, S = 11, T= 000, C = 001 decode the string 001110000100110000.

Thus, the efficient method of representation should use codewords such that:

1. The codewords have the prefix property, and
2. The symbols used frequently have shorter codewords than those used less often.

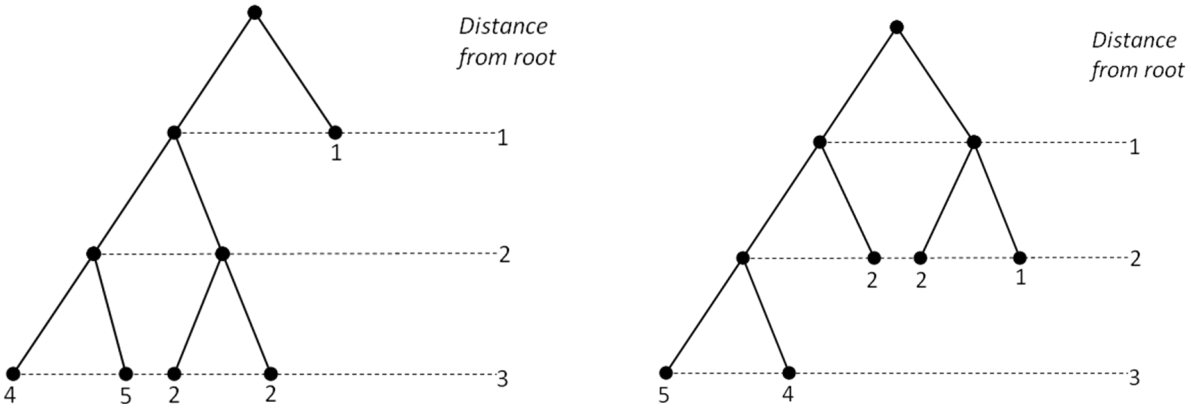
Any binary tree can be used to construct a set of codewords with the prefix property, by assigning 0 to each edge from a parent to its left child, and 1 to each edge from a parent to its right child. Following the unique path from the root to a terminal vertex (leaf) will give a string of 0s and 1s.

Example 3. For the binary tree, bellow, label the edges and find the corresponding codewords.



*A **binary tree for the weights** $w_1, w_2, \ldots, w_k$ is a binary tree with k leafs labeled $w_1, w_2, \ldots, w_k$. A binary tree for the weights $w_1, w_2, \ldots, w_k$ has **weight** $d_1w_1 + d_2w_2 + \ldots + d_kw_k$, where d_i is the length of the path from the root to the vertex labeled w_i (i=1, 2, …, k).*

Example 4. The figure bellow shows two binary trees, both with weights 1, 2, 2, 4 and 5. What is the weight of each tree?



Example 5. Consider using trees from the previous example to encode the text "ANNA AND DANNY". What is the length (in bits) of the compressed text?

For the coding problem, we want to find a binary tree of the smallest possible weight, in which the weights are the frequencies of the symbols to be encoded. A binary tree is called optimal binary tree for the weights $w_1, w_2, \ldots, w_k$ when its weight is as small as possible.

David A. Huffman (1925 – 1999), wrote a term paper in 1951 during his graduate work at MIT. The paper is the algorithm that constructs the optimal binary tree. Huffman coding is the fundamental algorithm in data compression. It is extensively used to compress bit strings representing text and it also plays an important role in compressing audio and image files.

The basic Algorithm:

1. Scan text to be compressed and count the occurrence of each character.
2. Sort or prioritize characters based on number of occurrences in text.
3. Build Huffman code tree based on prioritized list.
4. Perform a traversal of tree to determine all codewords.
5. Scan text again and create new file using the Huffman codes

Step 3 is explained in detail in the following algorithm.Building the tree

Create binary tree nodes with character and frequency of each character.
Place nodes in a priority queue. The lower the occurrence, the higher the priority in the queue.

While priority queue contains two or more nodes {
- ✓ Create new node
- ✓ Dequeue node and make it left subtree
- ✓ Dequeue next node and make it right subtree
- ✓ Calculate the frequency of new node as the sum of frequencies of left and right children.
- ✓ Enqueue new node back into queue

}

Example 6. Build Huffman tree for the short text "ANNA AND DANNY".

Frequency table:

Char	Frequency
A	4
N	5
space	2
D	2
Y	1

Initial Queue:

(continue on next page)

(continue Example 6.)

Build tree:

Lower bound for lossless compression.

Claude Shannon (1916 – 2001), "the father of information theory", derived a lower bound for any lossless compression.

Let n be a number of characters in a file, and let p_i be the probability that any given character in the file is character i. Then, lower bound on the number of bits per character that any encoding scheme would need is:

$$\sum_i^n p_i \log_2\left(\frac{1}{p_i}\right)$$

This important property is called the entropy.

Example 7. Consider the following table with normalized frequencies (i.e. probabilities)

Char	Normalized Frequency
A	0.5
B	0.25
C	0.25

Build Huffman tree and calculate the expected number of bits per character.

The lower bound on the number of bits per character is:

$$\sum_i^3 p_i \log_2\left(\frac{1}{p_i}\right) = 0.5\cdot\log_2\left(\frac{1}{0.5}\right) + 0.25\cdot\log_2\left(\frac{1}{0.25}\right) + 0.25\cdot\log_2\left(\frac{1}{0.25}\right) = 1.5$$

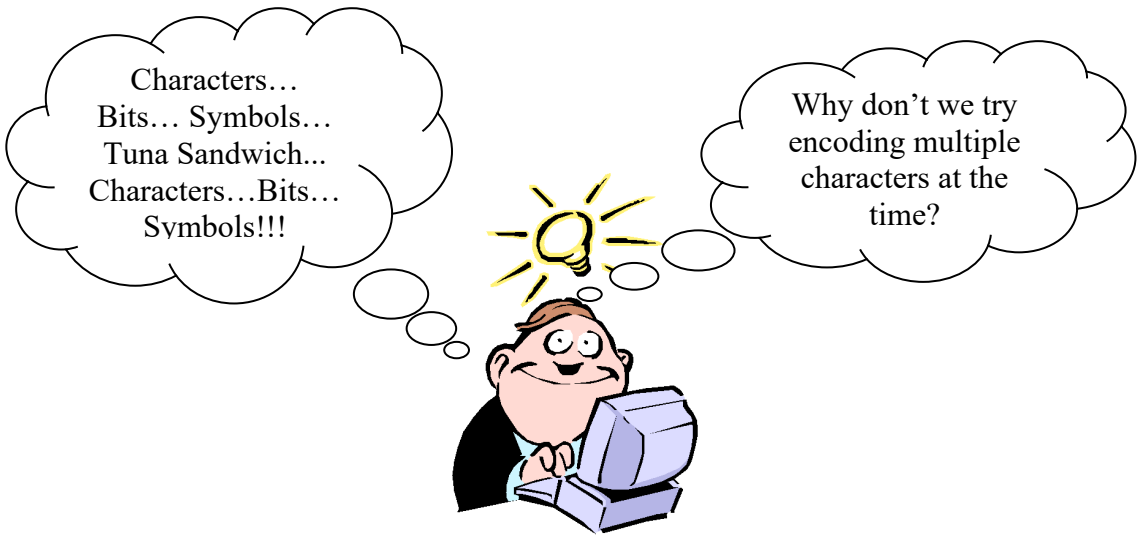
Example 8. Consider the following table with normalized frequencies (i.e. probabilities)

Char	Normalized Frequency
A	0.8
B	0.19
C	0.01

Build Huffman tree and calculate the expected number of bits per character.

The lower bound on the number of bits per character is:

$$\sum_{i}^{3} p_i \log_2\left(\frac{1}{p_i}\right) = 0.8 \cdot \log_2\left(\frac{1}{0.8}\right) + 0.19 \cdot \log_2\left(\frac{1}{0.19}\right) + 0.01 \cdot \log_2\left(\frac{1}{0.01}\right) = 0.779$$



Example 9. Consider the following table with normalized frequencies (i.e. probabilities)

Char	Normalized Frequency
A	0.8
B	0.19
C	0.01

The minimum number of bits per character is the same as in the previous example: 0.779

Build Huffman tree and calculate the expected number of bits per character based on the following probability distribution (obtained from the table above):

Symbol	Normalized Frequency
AA	0.6400
AB	0.1520
AC	0.0080
BA	0.1520
BB	0.0361
BC	0.0019
CA	0.0080
CB	0.0019
CC	0.0001