10. HUFFMANTREES

To encode a text that comprises symbols from some *n*-symbol alphabet by assigning to each of the text's symbols some sequence of bits called the ***code word***. For example, we can use a ***fixed- length encoding*** that assigns to each symbol a bit string of the same length *m* ($m \geq \log2\ n$).

This is exactly what the standard ASCII code does. ***Variable-length encoding***, which assigns code words of different lengths to different symbols, introduces a problem that fixed-length encoding does not have. Namely, how can we tell how many bits of an encoded text represent the first (or, more generally, the *i*th) symbol? To avoid this complication, we can limit ourselves to the so-called ***prefix-free*** (or simply ***prefix***) ***codes***.

In a prefix code, no code word is a prefix of a code word of another symbol. Hence, with such an encoding, we can simply scan a bit string until we get the first group of bits that is a code word for some symbol, replace these bits by this symbol, and repeat this operation until the bit string's end is reached.

Huffman Algorithm:

Step 1 Initialize *n* one-node trees and label them with the symbols of the alphabet given. Record the frequency of each symbol in its tree's root to indicate the tree's ***weight***. (More generally, the weight of a tree will be equal to the sum of the frequencies in the tree's leaves.)

Step 2 Repeat the following operation until a single tree is obtained. Find two trees with the smallest weight (ties can be broken arbitrarily, but see Problem 2 in this section's exercises). Make them the left and right subtree of a new tree and record the sum of their weights in the root of the new tree as its weight.

A tree constructed by the above algorithm is called a **Huffman tree**. It defines in the manner described above is called a **Huffman code**.

EXAMPLE Consider the five-symbol alphabet {A, B, C, D, _} with the following occurrence frequencies in a text made up of these symbols:

symbol	A	B	C	D	_
frequency	0.35	0.1	0.2	0.2	0.15

The Huffman tree construction for this input is shown in Figure 3.18

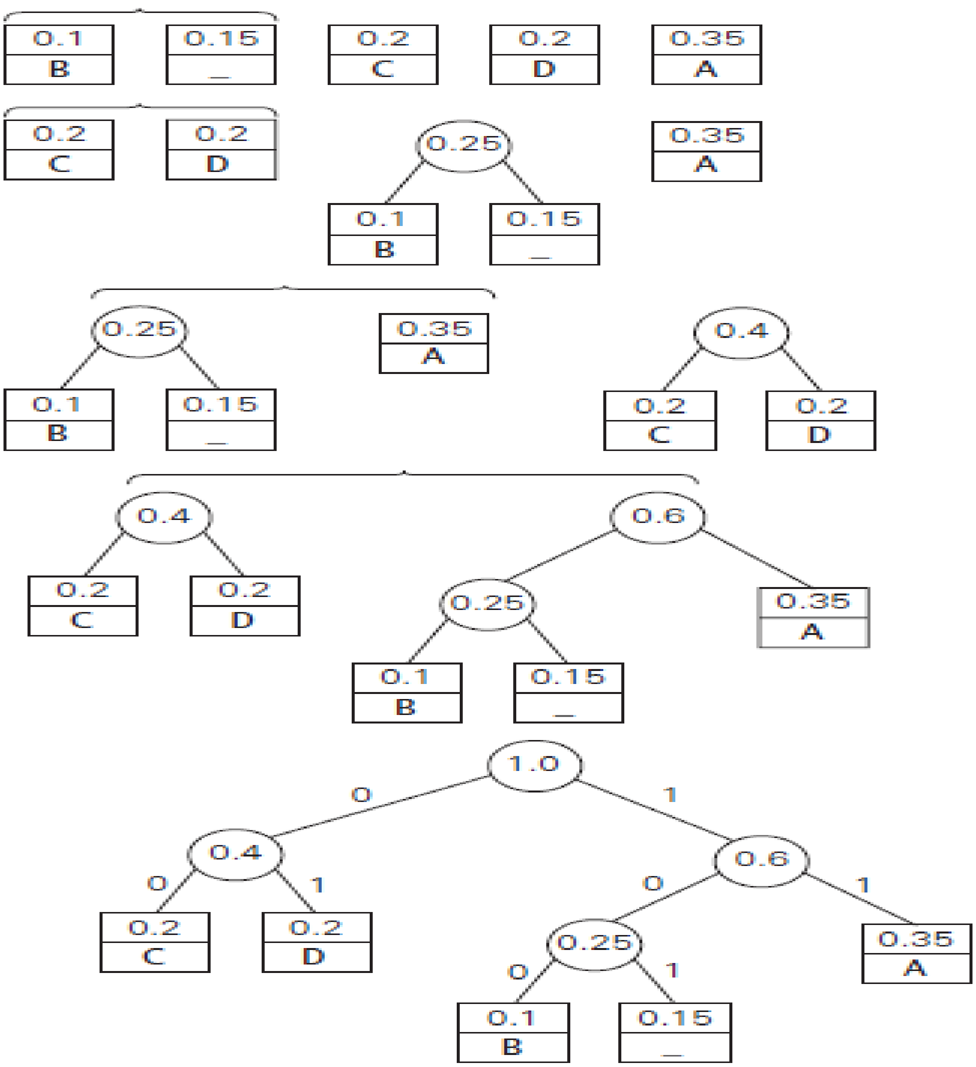

FIGURE 3.10.1 Example of constructing a Huffman coding tree.

The resulting code words are as follows:

symbol	A	B	C	D	_
frequency	0.35	0.1	0.2	0.2	0.15

Code word	11	100	00	01	101

Hence, DAD is encoded as 011101, and 10011011011101 is decoded as BAD_AD. With the occurrence frequencies given and the code word lengths obtained, the average number of bits per symbol in this code is 2 .0.35 + 3 .0.1+ 2 .0.2 + 2 .0.2 + 3 .0.15 =2.25.

We used a fixed-length encoding for the same alphabet, we would have to use at least 3 bits per each symbol. Thus, for this toy example, Huffman's code achieves the ***compression ratio*** **-** a standard measure of a compression algorithm's effectiveness of (3−2.25) / 3 · 100% = **25%**. In other words, Huffman's encoding of the text will use 25% less memory than its fixed-length encoding.

Running time is O(*n log n*), as each priority queue operation takes time O(*log n*).

Applications of Huffman's encoding

1. Huffman's encoding is a variable length encoding, so that number of bits used are lesser than fixed length encoding.
2. Huffman's encoding is very useful for file compression.
3. Huffman's code is used in transmission of data in an encoded format.
4. Huffman's encoding is used in decision trees and game playing.