Text Book: Introduction to the Design and Analysis of Algorithms

Author: Anany Levitin 2 nd Edition

Unit-4

## 9. Huffman Coding

### Huffman Trees

The Huffman trees are constructed for encoding a given text of n characters. While encoding a given text, each character is associated with some of bits called the *codeword*.

- Fixed length encoding: Assigns to each character a bit string of the same length.
- Variable length encoding: Assigns codewords of different lengths to different characters.
- Prefix free code: In Prefix free code, no codeword is a prefix of a codeword of another character.

### Binary prefix code :

- The characters are associated with the leaves of a binary tree.
- All left edges are labeled as 0, and right edges as 1.
- Codeword of a character is obtained by recording the labels on the simple path from the root to the character's leaf.
- Since, there is no simple path to a leaf that continues to another leaf, no codeword can be a prefix of another codeword.
- Huffman's algorithm achieves data compression by finding the best variable length binary encoding scheme for the symbols that occur in the file to be compressed. Huffman coding uses frequencies of the symbols in the string to build a variable rate prefix code

- Each symbol is mapped to a binary string
- More frequent symbols have shorter codes
- No code is a prefix of another code (prefix free code)

- Huffman Codes for data compression achieves 20-90% Compression.

**Huffman Algorithm:**

Input: Alphabet and frequency of each symbol in the text.

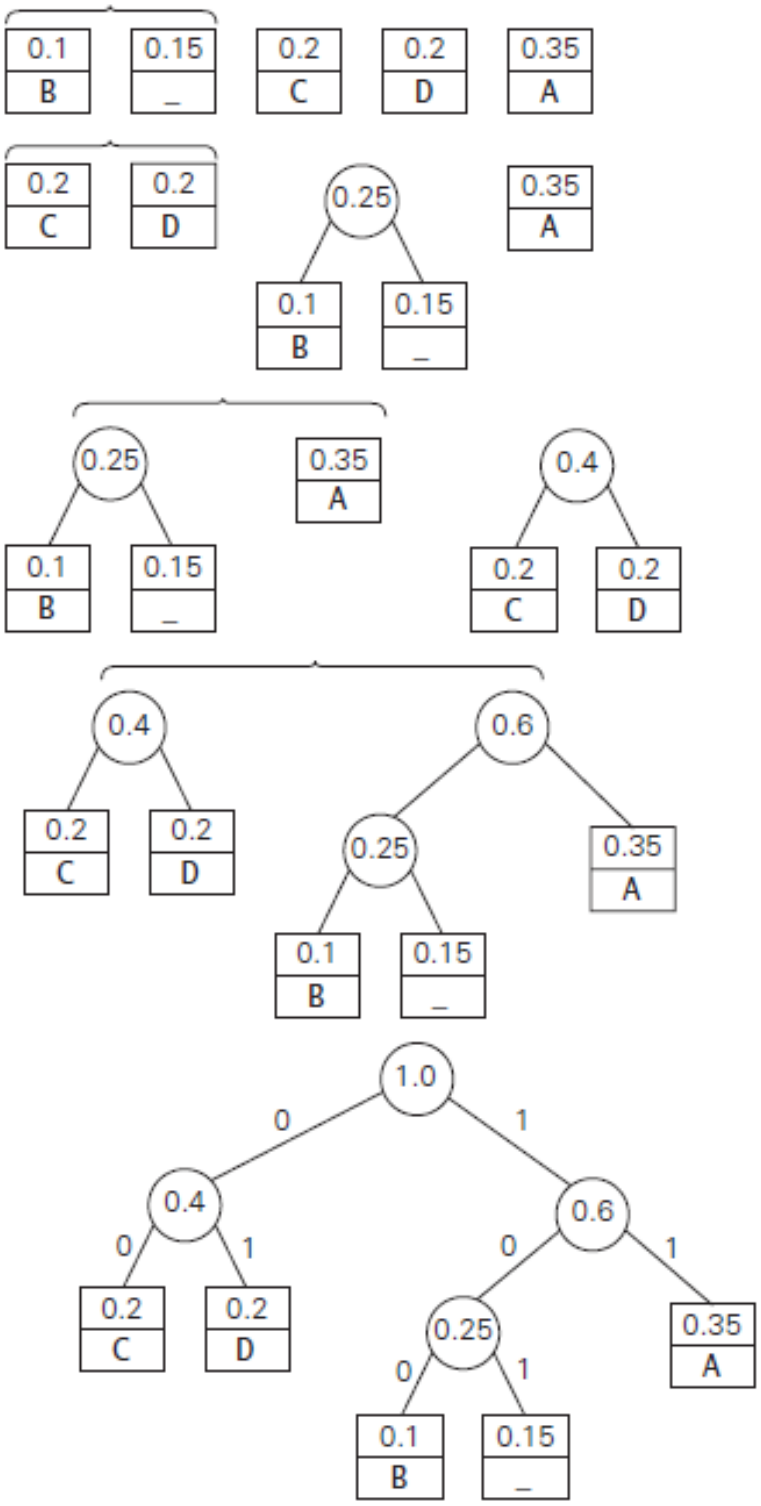
Step 1: Initialize n one-node trees (forest) and label them with the symbols of the alphabet given. Record the frequency of each symbol in its tree's root to indicate the tree's weight. (Generally, the weight of a tree will be equal to the sum of the frequencies in the tree's leaves.)

Step 2: Repeat the following operation until a single tree is obtained. Find two trees with the smallest weight. 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—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 |


0.1
B
0.15
_
0.2
C
0.2
D
0.35
A
0.25
0.4
0.6
1.0
0
1

The resulting codewords are as follows:

| symbol | A | B | C | D | _ |
|---|---|---|---|---|---|
| frequency | 0.35 | 0.1 | 0.2 | 0.2 | 0.15 |
| codeword | 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.

Had 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.

Huffman's encoding is one of the most important file-compression methods.

In addition to its simplicity and versatility, it yields an optimal, i.e., minimal-length encoding.