Char	Freq	Prob.	Range	CumFreq
		Total CumFreq=		10
S	5	5/10 = 0.5	[0.5, 1.0)	5
W	1	1/10 = 0.1	[0.4, 0.5)	4
I	2	2/10 = 0.2	[0.2, 0.4)	2
M	1	1/10 = 0.1	[0.1, 0.2)	1
␣	1	1/10 = 0.1	[0.0, 0.1)	0

Table 2.46: Frequencies and Probabilities of Five Symbols.

Char.		The calculation of low and high
S	L	$0.0 + (1.0 - 0.0) \times 0.5 = 0.5$
	H	$0.0 + (1.0 - 0.0) \times 1.0 = 1.0$
W	L	$0.5 + (1.0 - 0.5) \times 0.4 = 0.70$
	H	$0.5 + (1.0 - 0.5) \times 0.5 = 0.75$
I	L	$0.7 + (0.75 - 0.70) \times 0.2 = 0.71$
	H	$0.7 + (0.75 - 0.70) \times 0.4 = 0.72$
S	L	$0.71 + (0.72 - 0.71) \times 0.5 = 0.715$
	H	$0.71 + (0.72 - 0.71) \times 1.0 = 0.72$
S	L	$0.715 + (0.72 - 0.715) \times 0.5 = 0.7175$
	H	$0.715 + (0.72 - 0.715) \times 1.0 = 0.72$
␣	L	$0.7175 + (0.72 - 0.7175) \times 0.0 = 0.7175$
	H	$0.7175 + (0.72 - 0.7175) \times 0.1 = 0.71775$
M	L	$0.7175 + (0.71775 - 0.7175) \times 0.1 = 0.717525$
	H	$0.7175 + (0.71775 - 0.7175) \times 0.2 = 0.717550$
I	L	$0.717525 + (0.71755 - 0.717525) \times 0.2 = 0.717530$
	H	$0.717525 + (0.71755 - 0.717525) \times 0.4 = 0.717535$
S	L	$0.717530 + (0.717535 - 0.717530) \times 0.5 = 0.7175325$
	H	$0.717530 + (0.717535 - 0.717530) \times 1.0 = 0.717535$
S	L	$0.7175325 + (0.717535 - 0.7175325) \times 0.5 = 0.71753375$
	H	$0.7175325 + (0.717535 - 0.7175325) \times 1.0 = 0.717535$

Table 2.47: The Process of Arithmetic Encoding.

Char.	Code–low		Range
S	$0.71753375 - 0.5$	$= 0.21753375$	$/0.5 = 0.4350675$
W	$0.4350675 - 0.4$	$= 0.0350675$	$/0.1 = 0.350675$
I	$0.350675 - 0.2$	$= 0.150675$	$/0.2 = 0.753375$
S	$0.753375 - 0.5$	$= 0.253375$	$/0.5 = 0.50675$
S	$0.50675 - 0.5$	$= 0.00675$	$/0.5 = 0.0135$
␣	$0.0135 - 0$	$= 0.0135$	$/0.1 = 0.135$
M	$0.135 - 0.1$	$= 0.035$	$/0.1 = 0.35$
I	$0.35 - 0.2$	$= 0.15$	$/0.2 = 0.75$
S	$0.75 - 0.5$	$= 0.25$	$/0.5 = 0.5$
S	$0.5 - 0.5$	$= 0$	$/0.5 = 0$

Table 2.49: The Process of Arithmetic Decoding.

Huffman Coding

1. Build a binary tree where the leaves of the tree are the symbols in the Alphabet.
2. The edges of the tree are labeled by a 0 or 1.
3. Derive the Huffman code from the Huffman tree.

Starts by building a list of all alphabet symbols in descending order of their probabilities. It then constructs a tree with symbols at every leaf, from the bottom

up. This is done in steps. At each step the two symbols with smallest probabilities are selected, added to the top of the partial tree, deleted from the list, and replaced with an auxiliary symbol representing the two original symbols. When the list is reduced to just one auxiliary symbol (representing the entire alphabet), the tree is complete. The tree is then traversed to determine the codes of the symbols [1]. The major disadvantage of this procedure is, that the information about the Huffman tree has to be embedded into the compressed files or data transmissions. A code table or the symbol's frequencies must be part of the header data.

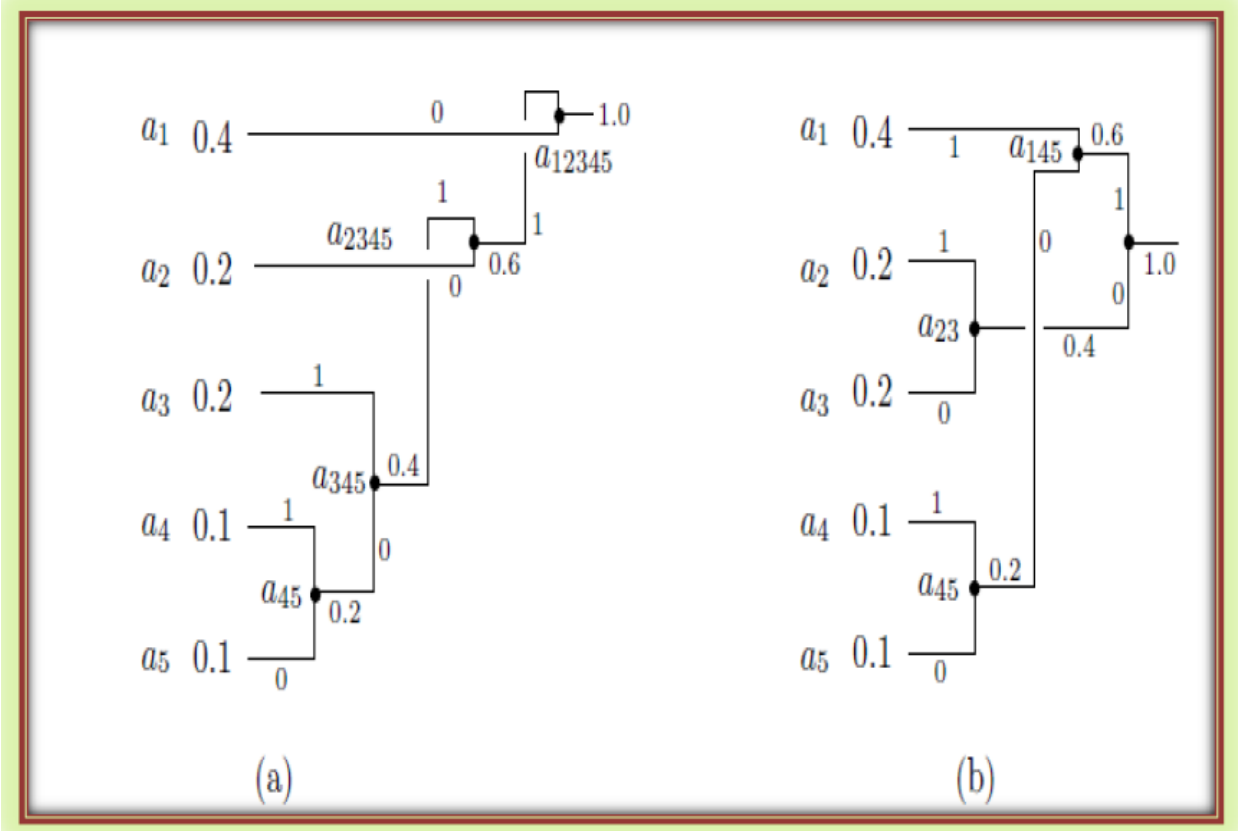

Figure (1): Huffman Code-Binary trees

It is shown in Figure (1- a) "lying on its side" with its root on the right and its five leaves on the left. To assign the codes, we arbitrarily assign a bit of 1 to the top edge, and a bit of 0 to the bottom edge, of every pair of edges. This results in the codes 0, 10, 111, 1101, and 1100. The assignment of bits to the

edges is arbitrary. Figure (1-b) shows how the same five symbols can be combined differently to obtain a different Huffman code (11, 01, 00, 101, and 100).

In the tree of Figure 1-a, symbol *a45* is combined with *a*3, whereas in the tree of 1-b it is combined with *a*1. The rule is: **When there are more than two smallest-probability nodes, select the ones that are lowest and highest in the tree and combine them.** This will combine symbols of low probability with ones of high probability, thereby reducing the total variance of the code.

 The average size of code of Figure 1-a is:

$$0.4 \times 1 + 0.2 \times 2 + 0.2 \times 3 + 0.1 \times 4 + 0.1 \times 4 = 2.2 \text{ bits/symbol}$$

Figure 1-b shows how the same five symbols can be combined differently to obtain a different Huffman code (11, 01, 00, 101, and 100).

The average size of code of Figure 1-b is:

$0.4 \times 2 + 0.2 \times 2 + 0.2 \times 2 + 0.1 \times 3 + 0.1 \times 3 = 2.2$ bits/symbol

 The best code is the one with the smallest variance. The variance of a code measures how much the sizes of the individual codes deviate from the average size.

The variance of code of Figure 1-a is:

$$0.4(1 - 2.2)2 + 0.2(2 - 2.2)2 + 0.2(3 - 2.2)2 + 0.1(4 - 2.2)2 + 0.1(4 - 2.2)2 = 1.36$$

The variance of code of Figure 1-b is:

$$0.4(2 - 2.2)2 + 0.2(2 - 2.2)2 + 0.2(2 - 2.2)2 + 0.1(3 - 2.2)2 + 0.1(3 - 2.2)2 = 0.16.$$

So, Code of Figure 1-b is therefore preferable.