Corollary 28 *Consider a coding from a length n vector of source symbols, $\boldsymbol{x} = (x_1 x_2 \dots x_n)$, to a binary codeword of length $\ell(\boldsymbol{x})$. Then the average codeword length per source symbol for an optimal prefix-free code satisfy*

$$\frac{1}{n}H(X_1 X_2 \dots X_n) \leq L \leq \frac{1}{n}H(X_1 X_2 \dots X_n) + \frac{1}{n}$$

where $L = \frac{1}{n}E[\ell(\boldsymbol{x})]$. □

Letting n approaching infinity this will sandwich the length at $\frac{1}{n}H(\boldsymbol{X}) \to H_\infty(X)$. Expressed more formally we get the following corollary.

Corollary 29 *If $X_1 X_2 \dots X_n$ is a stationary stochastic process, the average codeword length for an optimal binary prefix-free code*

$$L \to H_\infty(X), \quad n \to \infty$$

where $H_\infty(X)$ is the entropy rate for the process □

5.4 Huffman Codes

In the previous section it was seen that the Shannon-Fano code construction gives a code with average codeword length bounded by

$$H_D(X) \leq L < H(X) + 1$$

but also that it is not necessary an optimal code construction. By optimal code we here mean a code with the minimum average length over all prefix-free codes for the source. Stated as a definition we get the following.

Definition 19 *An optimal prefix-free code is a prefix-free code that minimizes the expected codeword length*

$$L = \sum_i p(x_i)\ell_i$$

over all prefix-free codes. □

In this section we will introduce a code construction due to David Huffman [8]. It was first developed by Huffman as part of a class assignment during the first ever course in Information Theory, given by Robert Fano at MIT. We will here first state the algorithm for the binary case and then show that the procedure generates an optimal code. The algorithm will also be generalized for D-ary codes.

The idea of the Huffman code construction is to list all letters of $\mathcal{X}$ as nodes. Then iteratively find the two least probable nodes and merge in a binary tree, let the rot represent a new node instead of two merged. Written as an algorithm we get the following.

Algorithm 1 (Binary Huffman code)
To construct the code tree:

1. *Sort the symbols according to their probabilities.*
2. *Let x_i and x_j, with probabilities p_i and p_j, respectively, be the two least probable symbols*
   - *Remove them from the list and connect them in a binary tree.*
   - *Add the root node $\{x_i, x_j\}$ as one symbol with probability $p_{ij} = p_i + p_j$ to the list.*
3. *IF one symbol in the list*
   STOP
   ELSE
   GOTO 1

Before we go into details of why this algorithm works we will get acquainted with it through a couple of example. The first one is a small example that we will go through in detail, and then follows a slightly bigger.

Example 35 The random variable X is drawn from $\mathcal{X} = \{x_1, x_2, x_3, x_4\}$ with the probabilities

x	x_1	x_2	x_3	x_4
$p(x)$	$1/2$	$1/4$	$1/8$	$1/8$

In Figure 5.10(a) each of the symbols in $\mathcal{X}$ represent a node. To start the algorithm find the two nodes with least probabilities. In this case it is the nodes x_3 and x_4 with probability $1/8$ each. Merge these nodes in a binary tree and add the root as a node to the list. Now, the list contains three nodes, x_1, x_2 and x_3x_4, where the last one represents the newly added tree. This list is shown Figure 5.10(b). since there are more than one node left continue from the beginning and find the two least probable nodes. Now it is the nodes x_2 and x_3x_4 that should be merged, see Figure 5.10(c). There are now two nodes left, x_1 and $x_2x_3x_4$ and they represent he two least probable nodes. Merge and let the root represent a new node to get Figure 5.10(d). Now it is only one node left and we should stop. Redrawing the tree and setting up the coding table we get the tree and the code table shown in Figure 5.11

The average codeword length in the constructed code is, according to the path length lemma,

$$L = 1 + \frac{1}{2} + \frac{1}{4} = 1.75$$

From Theorem 26 we know that the codeword length is lower bounded by the entropy of the source. So, to compare we derive the entropy as

$$H\left(\tfrac{1}{2}, \tfrac{1}{4}, \tfrac{1}{8}, \tfrac{1}{8}\right) = 1.75$$

Since we have equality between the codeword length and the entropy we can directly say that the constructed code is optimal.

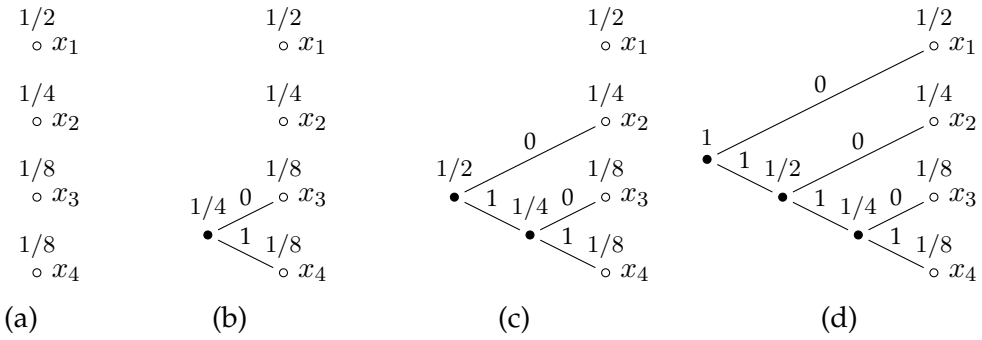
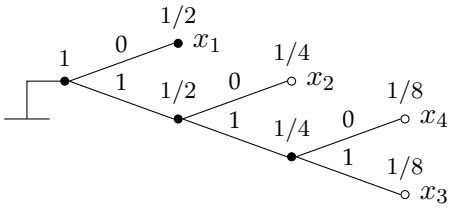


Figure 5.10: Construction of the Huffman tree.



x	$p(x)$	y
x_1	1/2	0
x_2	1/4	10
x_3	1/8	110
x_4	1/8	111

Figure 5.11: The tree and the code table for the Huffman code in Example .

The next example is a little bit bigger and shows that there is not always equality in the lower bound for optimal codeword length.

Example 36 The random variable X is drawn from $\mathcal{X} = \{x_1, x_2, \ldots, x_6\}$ with the probabilities

x	x_1	x_2	x_3	x_4	x_5	x_6
$p(x)$	0.05	0.10	0.15	0.20	0.23	0.27

In Figure 5.12(a) the Huffman tree is constructed. The labeling of the inner nodes represent the order in which they are constructed in the algorithm. In Figure 5.12(b) the same tree redrawn to better see the structure.

The code can be written in a table as

x	x_1	x_2	x_3	x_4	x_5	x_6
y	0000	0001	001	10	11	01

The average codeword length can be calculated as

$$L = 1 + 0.57 + 0.3 + 0.15 + 0.43 = 2.45$$

and the entropy of the source is

$$H(X) = H(0.05, 0.10, 0.15, 0.20, 0.23, 0.27) = 2.42$$

In he previous example the Huffman code does not give equality in the lower bound for the average length. In the following we will show that in fact the procedure gives an

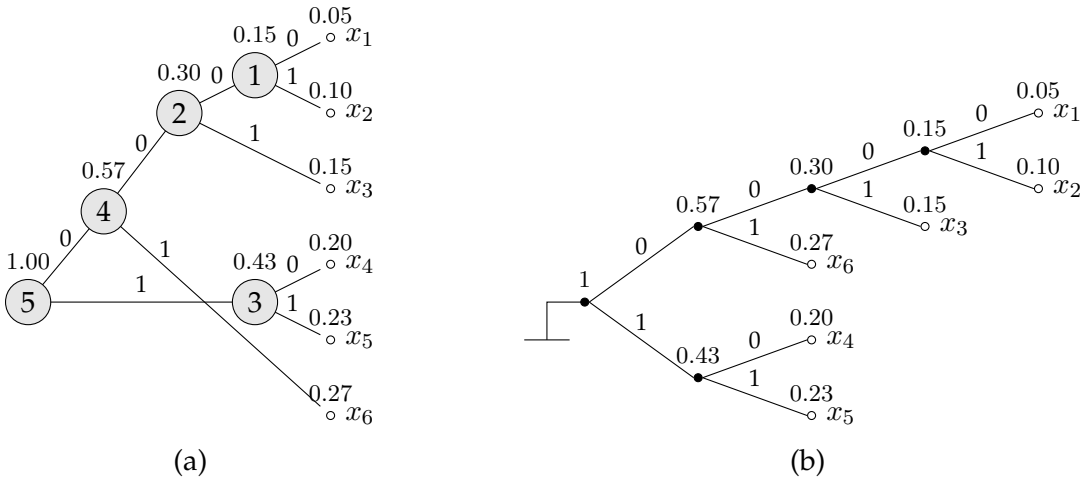


Figure 5.12: Construction of the Huffman tree.

optimal code. In the previous example that means there are no other prefix-free code that gives equality in the bound either.

To show that Huffman codes are are optimal we first set up a couple of rules that must apply to an optimal code. Then, it is noted that these rules boils down to the algorithm given by Huffman, which proves that the codes are optimal.

First we notice that in an optimal code the lengths for the codewords must follow the probabilities for the source symbols. That is, codewords corresponding to source symbols with low probability should not be shorter than codewords corresponding to more probable source symbols. Intuitively it is clear that if we have a codeword which length is shorter than a codeword with higher probability, we will get a shorter average length if we swap the codewords. To derive this we assume that the source symbols x_i and x_j have the probabilities p_i and p_j, respectively, where $p_i < p_j$. The corresponding codewords have the lengths ℓ_i and ℓ_j, respectively, and we assume that $\ell_i \leq \ell_j$. We will now see that by swapping the lengths we will decrease the average length,

$$\begin{aligned} L &= \sum_k p_k \ell_k = \sum_{k \neq i,j} p_k \ell_k + p_i \ell_i + p_j \ell_j \\ &= \sum_{k \neq i,j} p_k \ell_k + p_i \ell_i + p_j \ell_j + p_i \ell_j + p_j \ell_i - p_i \ell_j - p_j \ell_i \\ &= \underbrace{\sum_{k \neq i,j} p_k \ell_k + p_i \ell_j + p_j \ell_i}_{L'} + \underbrace{(p_j - p_i)}_{>0} \underbrace{(\ell_j - \ell_i)}_{\geq 0} > L' \end{aligned}$$

where L' is the length when the codewords are swapped. This shows that by swapping the codewords we get $L > L'$. Hence, in an optimal code codewords corresponding to less probable symbols are not shorter than codewords corresponding to more probable symbols.

Next we use the tree representation of a prefix-free code to get the following lemma.

Lemma 30 *There exists an optimal prefix-free binary code such that*

1. *The corresponding binary tree has no unused end-nodes (leaves).*
2. *The two least probable codewords differ only in the last bit.*

□

The first part can be seen from an assumption that there is an optimal with an unused leaf in the tree. Then the predecessor node of this leaf has only one branch. By removing this last branch and placing the leaf at the predecessor node we obtain a shorter codeword, and a shorter average length. Hence, the assumption that the code was optimal failed.

Assume that the least probable symbol has a codeword according to $y_1 \dots y_{\ell-1}0$. Then since there are no unused leaves, we also know that there exists a codeword $y_1 \dots y_{\ell-1}1$. If this is not the second least probable codeword, the second least probable codeword has the same length. So, we can swap this codeword with $y_1 \dots y_{\ell-1}1$ and we get the desired result, without any change in the average length.

We can now use the above properties to find a way to construct an optimal code. If we have a random variable X with n different outcomes we should construct a binary code tree with n leaves (since we know there are no unused leaves). Then, we can assume that the two least probable codewords corresponding to the symbols x_n and x_{n-1}, occur with probabilities p_n and p_{n-1}, respectively. According to the above we can also assume that these two codewords differs only in the last digit, see Figure 5.13. Then the preceding intermediate node for those two leaves has the probability

$$\tilde{p}_{n-1} = p_n + p_{n-1}$$

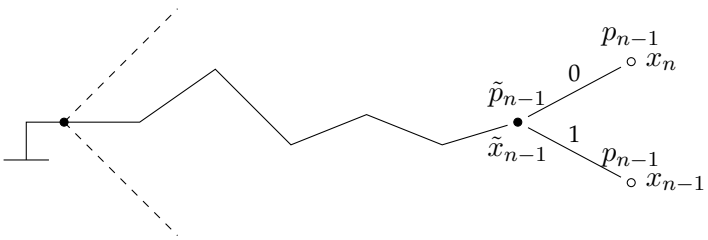


Figure 5.13: A code tree for an optimal code.

A new code tree with $n-1$ leaves can now be constructed by replacing the codewords for x_n and x_{n-1} by its intermediate node, which we call $\tilde{x}_{n-1}$ in the figure. denote by L the average length in the original code with n codewords and $\tilde{L}$ the average length in the new code. From the path length lemma we get that

$$L = \tilde{L} + \tilde{p}_{n-1} = \tilde{L} + (p_n + p_{n-1})$$

From this we see that the code with n codewords can only be optimal if the code with $n-1$ elements is optimal. Continuing this reasoning until there are only two codewords left we can easily construct an optimal code using the two codewords 0 and 1. If we consider the steps we have taken here to construct an optimal code, it is exactly the same steps used in the algorithm for Huffman code. Hence we can conclude that the Huffman code is optimal.

Theorem 31 *A binary Huffman code is an optimal (prefix-free) code.* □

The construction according to Huffman will give an optimal code. But there are also optimal codes that does not follow the Huffman principles. For example, the requirement that the two least likely codewords differ only in the last symbol must not be fulfilled in an optimal code. We can exchange one of them with another codeword with the same length. Then the overall length is not changed and the code still optimal.

It can also be that the Huffman procedure can create several different codes. In the next example two Huffman codes can be created for the same distribution is shown.

Example 37 The random variable X is drawn from $\mathcal{X} = \{x_1, x_2, \ldots, x_5\}$ with the probabilities

x	x_1	x_2	x_3	x_4	x_5
$p(x)$	0.4	0.2	0.2	0.1	0.1

The Huffman algorithm can produce two different trees, and thus two different codes, for this statistics. In Figure 5.14 the two trees are shown. The difference in the construction comes after the first step in the algorithm. Then there are three nodes with least probability, x_2, x_3 and x_1x_2. In the first alternative the nodes x_3 and x_1x_2 are merged into $x_3x_4x_5$ with probability 0.4, and in the second alternative the two nodes x_2 and x_3 are merged to x_2x_3.

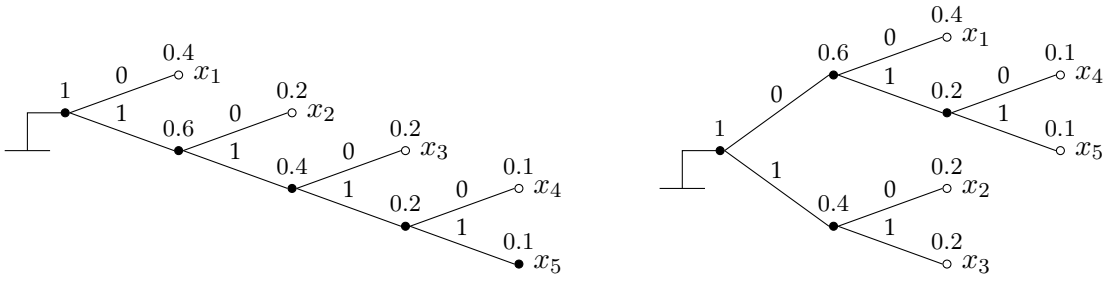


Figure 5.14: Two alternative Huffman trees for one source.

Calculating the average codeword length for the two alternatives will both give the same calculation

$$L_1 = L_2 = 1 + 0.6 + 0.4 + 0.2 = 2.2\text{bit}$$

In the example it is seen that both codes give the same (optimal) codeword lengths. However, the difference can be of important from another perspective. The source coding gives variations in the length of the coded symbols, i.e. the rate of the symbol varies from the encoder. In most communication schemes the transmission is done with a fixed maximum rate. To be able to handle this mismatch the transmitter and receiver is often equipped with buffers. At the same time we want the delays in the system to kept as

small as possible, and therefore the buffer sizes should also be small. This implies that the variations in the rates from the source encoder should be as small as possible. In the first alternative of the code tree the variation in length is much larger than in the second alternative. This will be reflected in the variations in the rates of the code symbol. One way to construct *minimum variance Huffman codes* is to always merge the shortest sub-trees when there is a choice. In the example Alternative 2 is a minimum variance Huffman code, and might be preferable to the first alternative.

5.4.1 D-ary Huffman code

So far only binary Huffman codes has been considered. In most applications this is enough but there are also cases when a larger alphabet is required. In this case we want to consider D-ary Huffman codes. The algorithm and the theory is in many aspects very similar. Instead of a binary tree we consider a D-ary tree. Such a tree with depth 1 has D leaves. Then it can be expanded by taking one leaf and adding D children. In the original tree one leaf is now an internal node and there are D new leaves, so there are now $D-1$ new leaves. Every following expansion will also give $D-1$ new leaves. Hence we get the following lemma.

Lemma 32 *The number of end-nodes in a D-ary tree is*

$$D + q(D-1)$$

for some integer q. □

A corresponding statement as in Lemma 30 for D-ary codes can be stated as the next lemma.

Lemma 33 *There are at most $D-2$ unused end-nodes in the tree for an optimal D-ary prefix-free code. They are all located at maximum depth.*

There exists an optimal D-ary prefix-free code where all unused end-nodes stems from the same mother node. □

The optimal code construction for the D-ary case can be shown in a similar way as for the binary case. It will then end up in the algorithm described next.

Algorithm 2 (D-ary Huffman code)
To construct the code tree:

1. *Sort the symbols according to their probabilities.*
   Fill up with zero probable nodes so that $N = D + q(D-1)$.
2. *Connect the D least probable symbols in a D-ary tree and remove them from the list. Add the root of the tree as a symbol.*

3. IF one symbol in the list
STOP
ELSE
GOTO 1

We show the procedure with an example.

Example 38 We want to construct an optimal code with $D03$ for the random variable X drawn from $\mathcal{X} = \{x_1, x_2, \ldots, x_6\}$ with the probabilities

x	x_1	x_2	x_3	x_4	x_5	x_6
$p(x)$	0.27	0.23	0.20	0.15	0.10	0.05

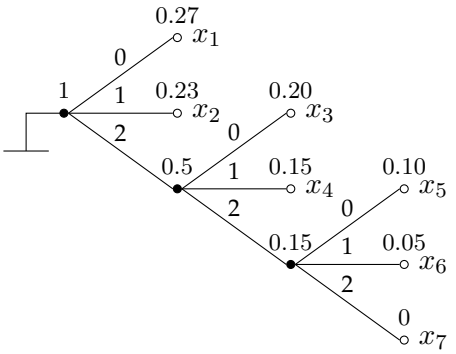
First we need to find the number of unused leaves in the tree. We know that we will have $N = D + q(D-1)$ leaves in the tree. That is, we can derive the least integer q as

$$q = \left\lceil \frac{N-D}{D-1} \right\rceil = \left\lceil \frac{6-3}{3-1} \right\rceil = \left\lceil \frac{3}{2} \right\rceil = 2$$

Hence, the number of leaves becomes

$$N = 3 + 2(3-2) = 7$$

Since there are only 6 codewords used we will have one unused leaf in the tree. To get this in the algorithm we add one symbol x_7 with probability 0. Then we can construct the code tree and the corresponding table as in Figure 5.15. The brances are labeled with the code alphabet $\{0, 1, 2\}$.



x	y
x_1	0
x_2	1
x_3	20
x_4	21
x_5	220
x_6	221

Figure 5.15: A 3-ary tree for a Huffman code.

The average length can still be calculated with the path length lemma,

$$L = 1 + 0.5 + 0.15 = 1.65$$

As a comparison, the lower bound of the length is the entropy,

$$H_3(X) = \frac{H(X)}{\log 3} \approx \frac{2.42}{1.59} = 1.52$$

We see that we do not reach the entropy. Still, this is an optimal code and it is not possible to find a prefix-free code with lower length.

Instead of the deribvation in the previous example to find the number of unused leaves in the tree, it is possible to derive it directly

To be done.

Lemma 34 *The number of unused end-nodes in the tree for an optimal D-ary prefix-free code with L codewords is*

$$D - t$$

where[4]

$$t = R_{D-1}(L-2) + 2$$

□

[4] $R_n(m)$ denotes the reminder when m is divided by n, i.e. the modulo operator.