For random processes we do not have i.i.d. variables. However, it is possible to generalize the theorem for this case, using the entropy rate. The convergence from the law of large numbers corresponding to (5.1) is based on the following. Let $\boldsymbol{x} = (x_1, x_2, \ldots, x_n)$ be a sequence from a stationary ergodic random process. Then

$$-\frac{1}{n}\log p(\boldsymbol{x}) \to H_\infty(X), \quad n \to \infty$$

in probability. The set of typical sequences should then be defined as sequences such as

$$\left|-\frac{1}{n}\log p(\boldsymbol{x}) - H_\infty(X)\right| \leq \varepsilon$$

This leads to the source coding theorem for ergodic sequences.

Theorem 22 *Let X^n be an stationary ergodic process. Then there exists a code which maps sequences $\boldsymbol{x}$ of length n into binary sequences such that the mapping is invertible and*

$$\frac{1}{n}E\Big[\ell(\boldsymbol{x})\Big] \leq H_\infty(X) + \varepsilon'$$

for sufficiently large n, where ε' can be made arbitrarily small. □

A random process is said to be ergodic if the statistically invariant over time, i.e. expectation, variance and other statistical measures does not change over time. For a formal proof we refer to e.g. [2].

5.3 Kraft inequality and optimal codeword length

So far we have not said anything on how the source code should be constructed. We have noted that the decoding should be the inverse function of the encoding, so the original message can be reconstructed. Next we will consider different classes of codes, where the level of decoding is the determining factor. After that an important inequality, due to Kraft [11], is introduced. The aim of this inequality is to set up a requirement for when it is possible to construct a certain type of code.

There are different classes of codes that can be used for compression. Below follows a list of the most important and how they are related. A code is said to be

- **Non-singular** if every $x \in \mathcal{X}$ maps to different codewords $x \in \mathcal{X}$, i.e. that the mapping is one to one. This implies that the distribution of X and Y are equivalent, and that $H(Y) = H(X)$.
- **uniquely decodable** if every finite vector $\boldsymbol{x} \in \mathcal{X}^k$ maps to different code vectors, i.e. the viewed as vectors the code

  $$C(x_1x_2\ldots x_k) = C(x_1)C(x_2)\ldots C(x_k) = y_1y_2\ldots y_k$$

  is non-singular. Such code is always decodable in a unique way. It might require to see the complete vector before the individual codewords can be decoded.

- **prefix-free** or **instantaneous** if no codeword is a prefix of another codeword. In such code each codeword can be decoded as soon as it is received.[1]

In Table 5.1 the probability density function for a random variable X is given. There are also five different examples of codes for this variable. The first code, $\mathcal{C}_1$ is an example of a direct binary mapping with equal lengths, not taking the distribution into account. With four source symbols we need at least $\log 4 = 2$ bits in a vector to have unique codewords. This can be used as a reference code representing the uncoded case. Since all codewords are equal in length, the average codeword length will also be $L(\mathcal{C}_1) = 2$.

x	$p(x)$	$\mathcal{C}_1$	$\mathcal{C}_2$	$\mathcal{C}_3$	$\mathcal{C}_4$	$\mathcal{C}_5$
x_1	$1/2$	00	0	0	0	0
x_2	$1/4$	01	1	0	01	10
x_3	$1/8$	10	01	1	011	110
x_4	$1/8$	11	10	11	0111	111

Table 5.1: Some different codes.

The second code in Table 5.1, $\mathcal{C}_2$ has short codewords for the probable symbols. This will decrease the average codeword length significantly,

$$L(\mathcal{C}_2) = \sum_x p(x)\ell(x) = 1.25 \text{ bit}$$

Considering the classifications above we see that the codewords are unique, meaning it is a non-singular code. However, considering the codeword sequence

$$\boldsymbol{y} = 00110\ldots$$

we encounter a problem in decoding. Since no codeword contain any double zeros it is easy to see that the first zero corresponds to the symbol x_1. But then the next pair of bits, 01, can either mean the combination x_1, x_2 or the single symbol x_3. There is no way that we can make a clear decision between the alternatives, which means the code is not uniquely decodable. The problem that has occurred is due to that we have unequal length in the codewords and that we cannot use any separator between the codewords. The unequal lengths is a requirements for having compression.

The third code, $\mathcal{C}_3$, is an example of a singular code, i.e. it is not a non-singular code. Here the two first codewords are the same. The average length is clearly improved, $L(\mathcal{C}_3) = 1.125$ but the price is that it is directly seen that it is not decodable.

The third code, $\mathcal{C}_4$, has unique codewords for all symbols, hence it is non-singular. Since all codewords starts with a zero, and this is the only occurrence of zeros, any code sequence can be uniquely decoded, implying the code is uniquely decodable. The only flaw of the code is that we cannot see the end of a codeword until we have seen the first symbol of the next codeword. This is due to that the code is not prefix-free. For example the codeword 0 is a prefix to all the other codewords, and 01 is a prefix to 011 and 0111. The average codeword length is $L(\mathcal{C}_4) = 1.875$.

[1] Often in the literature this type of code is called Prefix code. However this name is a bit misleading since there should be no prefixes in the code, and the name prefix-free gives a better understanding of the concept.

Finally, the last code, $\mathcal{C}_5$, is both non-singular and uniquely decodable. We can also, be inspection, see that the code is prefix-free. The fact that it is prefix-free gives an easy decoding rule. As soon as a codeword has been found in the sequence it can be decoded to the corresponding source symbol. For example if the code sequence is

$$\boldsymbol{y} = 01011010111\ldots$$

we start from the beginning and look for codewords. The first we find is 0 which corresponds to x_1. secondly we get 10 giving x_2, and after that 110 representing x_3. Continuing this we can decode the sequence as follows

$$\boldsymbol{y} = \underbrace{0}_{x_1}\underbrace{10}_{x_2}\underbrace{110}_{x_3}\underbrace{10}_{x_2}\underbrace{111}_{x_4}\ldots$$

From the above reasoning we can conclude that prefix-free codes are desirable since it is very easy to decode them. The class of uniquely decodable codes can also be decoded uniquely, but it might require that we have to consider the complete code sequence to start the decoding. Clearly, the class of prefix-free codes is a subclass of the uniquely decodable codes. One basic criteria for a code to be uniquely decodable is that the set of codewords is non-overlapping, that is the class of uniquely decodable codes is a subclass of the non-singular code. Furthermore, the class of non-singular codes is a subclass of all codes. In Figure 5.4 a graphical representation of the relation between the classes is shown.

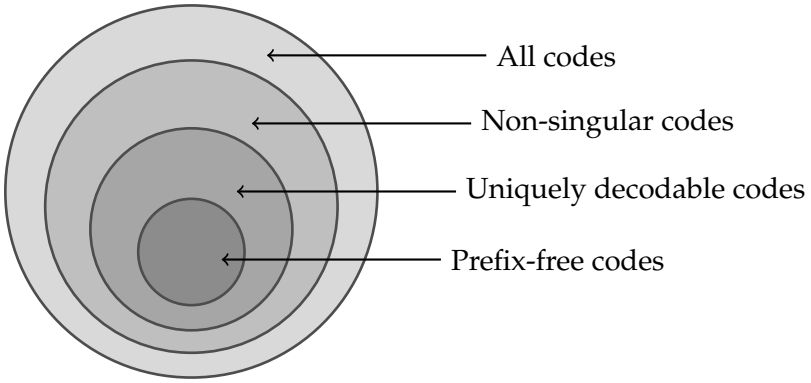


Figure 5.4: All prefix-free codes are uniquely decodable, and all uniquely decodable codes are non-singular.

In the continuation of this section we will consider prefix-free codes. For this analysis we first need to consider a tree structure. A general tree[2] has a root node which may have one or more child nodes. Each child node may also have one or more child nodes, and so on. A node that does not have any child nodes is called a leaf. In this text we will consider a D-ary tree, which means that each node has either zero or D child nodes. In Figure 5.5 two examples of D-ary trees are shown, the left with $D = 2$ and the right with $D = 3$. Notice that the trees grow to the right from the root. Normally, in computer science trees grow downwards, but in many topics related to information theory and communication theory they are drawn from left to right.

The depth of a node is the number of branches from the root note it is located in the tree. Starting with the root node it has depth 0. In the left tree of Figure 5.5 the node labeled A

[2]Even more general, in graph theory i tree is a directed graph where two nodes are connected by exactly one path.

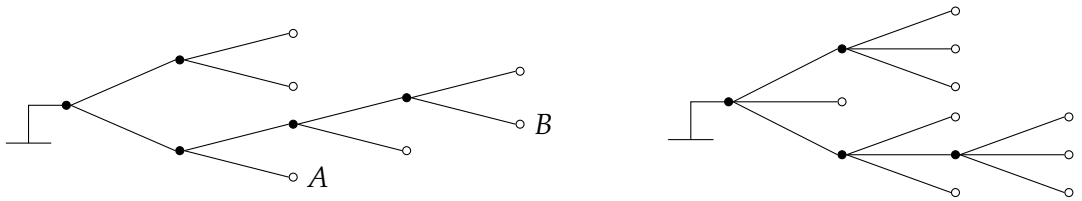


Figure 5.5: Examples of a binary ($D = 2$) and a 3-ary tree.

has depth 2 and the node labeled B has depth 4. A tree is said to be **full** if all leaves are located at the same depth. In Figure 5.6 a D-ary tree with $D = 2$ and depth 3 is shown. In a full D-ary tree of depth d there are D^d leaves. In the tree of Figure 5.6 there are $2^3 = 8$ leaves.

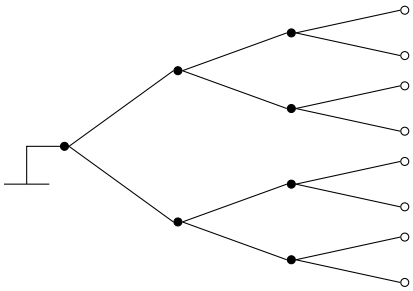


Figure 5.6: Example of a full binary tree of depth 3.

A prefix-free codes of alphabet size D can be represented in a D-ary tree. The first letter of the codeword is represented by a branch stemming from the root. The second letter is represented by a branch stemming from a node at depth 1, and so on. The end of a codeword is reached when there are no children, i.e. a leaf is reached. In this way a structure is built where each sequence in the tree cannot be a prefix of another codeword. In Figure 5.7 the prefix-free code $\mathcal{C}_5$ from Table 5.1 is shown in a binary tree representation. n this representation the probabilities for each source symbol is also added. The labeling in the tree nodes is the sum of the probabilities for the source symbols stemming from that node, i.e. the probability that a codeword goes through that node. Among the codes in Table 5.1, the reference code $\mathcal{C}_1$ is also a prefix-free code. In Figure 5.8 a representation of this code is shown in a tree. Since all codewords are of equal length we get a full binary tree.

x	$p(x)$	$\mathcal{C}_5$
x_1	$1/2$	0
x_2	$1/4$	10
x_3	$1/8$	110
x_4	$1/8$	111

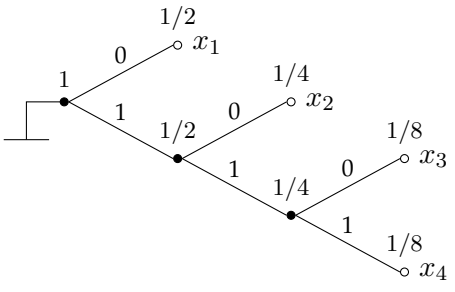


Figure 5.7: Representation of the prefix-free code $\mathcal{C}_5$ in a binary tree.

There are many advantages with the tree representation. One is that we get a graphi-

x	$p(x)$	$\mathcal{C}_1$
x_1	$1/2$	00
x_2	$1/4$	01
x_3	$1/8$	10
x_4	$1/8$	11

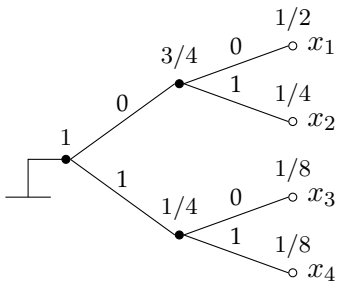


Figure 5.8: Representation of the code $\mathcal{C}_1$ in a (full) binary tree.

cal interpretation of the code, which is in many occasions a great help for our intuitive understanding. Another advantage is that there is an alternative way to calculate the average codeword length, formulated in the next lemma.

Lemma 23 (Path length lemma) *In a tree representation of a prefix-free code, the average codeword length $E[\ell(x)]$ equals the sum of probabilities for the inner nodes, including the root.* □

We will not give a proof for the lemma, instead an example following the same outline as a proof would do.

Example 31 Consider again the prefix-free code $\mathcal{C}_5$. The average codeword length can be derived as

$$L = \sum_x p(x)\ell(x) = \tfrac{1}{2}1 + \tfrac{1}{4}2 + \tfrac{1}{8}3 + \tfrac{1}{8}3 = 1.75$$

The derivations can be rewritten as

$$\begin{aligned} L &= \underbrace{\tfrac{1}{2}}_{x_1} + \underbrace{\tfrac{1}{4}+\tfrac{1}{4}}_{x_2} + \underbrace{\tfrac{1}{8}+\tfrac{1}{8}+\tfrac{1}{8}}_{x_3} + \underbrace{\tfrac{1}{8}+\tfrac{1}{8}+\tfrac{1}{8}}_{x_4} \\ &= \underbrace{\tfrac{1}{2}+\tfrac{1}{4}+\tfrac{1}{8}+\tfrac{1}{8}}_{1} + \underbrace{\tfrac{1}{4}+\tfrac{1}{8}+\tfrac{1}{8}}_{1/2} + \underbrace{\tfrac{1}{8}+\tfrac{1}{8}}_{1/4} = 1.75 \end{aligned}$$

By rearranging the terms of the sum it can be seen that each leaf probability is present in each of the nodes on the path to the leaf. Hence, by summing over all inner node probabilities the contribution from a leaf probability is the probability times the depth, or $p(x)\ell(x)$.

We are now ready to state and show a famous result, first published in a Master of Science thesis by Leon Kraft in 1949 [11]. It gives a requirement on the codeword lengths that can be used to form a prefix-free code. The result has been generalized by Brockway McMillan in 1956 to yield all uniquely decodable codes [15]. The result is often called Kraft inequality or Kraft-McMillan inequality.

Theorem 24 (Kraft inequality) *There exists a **prefix-free** D**-ary code** with codeword lengths $\ell_1, \ell_2, \ldots, \ell_k$ if and only if*

$$\sum_{i=1}^{k} D^{-\ell_i} \leq 1$$

□

To show this we consider a D-ary prefix-free code where the longest codeword length is $\ell_{\max} = \max_x \ell(x)$. This code can be represented in a D-ary tree. A full D-ary tree of depth $\ell_{\max}$ has $D^{\ell_{\max}}$ leaves. Use this tree to represent the codeword for the symbol x_i. This codeword is of length ℓ_i. Since it should be a prefix-free code the sub-tree spanned with x_i as a root is not allowed to used by any other codeword, hence it should be removed from the tree, see Figure 5.9. So, when inserting the codeword for symbol x_i we delete $D^{\ell_{\max}-\ell_i}$ leaves in the tree. Since we cannot remove more leaves than the tree has from the beginning we get that for a prefix-free code

$$\sum_i D^{\ell_{\max}-\ell_i} \leq D^{\ell_{\max}}$$

By canceling $D^{\ell_{\max}}$ on both sides, this proves that for a prefix-free code we have

$$\sum_i D^{-\ell_i} \leq 1$$

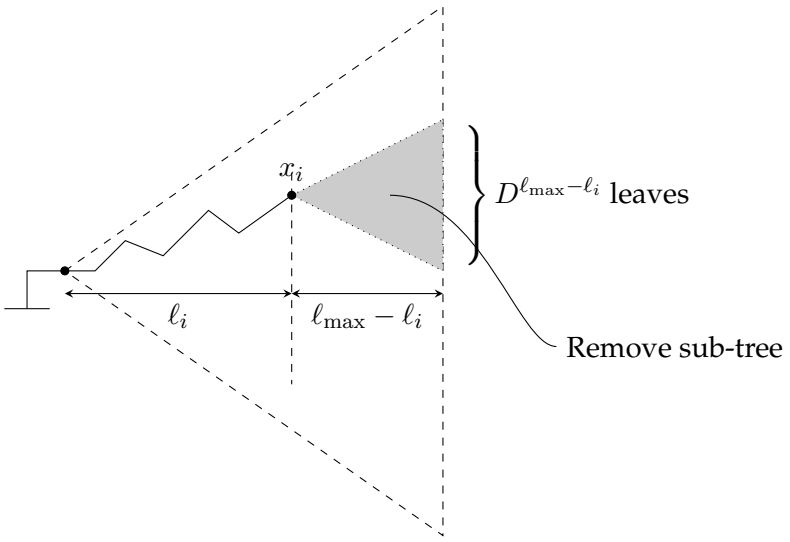


Figure 5.9: Tree construction for a general prefix-free code.

To show that if the inequality is fulfilled we can construct a prefix-free code we start by assuming that $\sum_i D^{\ell_i} \leq 1$ and that the codeword lengths are ordered, $\ell_1 \leq \ell_2 \leq \cdots \leq \ell_k$, where $\ell_k = \ell_{\max}$. Then we use the same construction as above. Start with the shortest codeword and remove the corresponding sub-tree. After $i < k$ steps the number of leaves left is

$$D^{\ell_{\max}} - \sum_{n=1}^{i} D^{\ell_{\max}-\ell_i} = D^{\ell_{\max}}\Big(1 - \sum_{n=1}^{i} D^{-\ell_i}\Big) > 0$$

where the last inequality comes from the assumption, i.e. as long as $i < k$ we have $\sum_{n=1}^{i} D^{-\ell_i} < 1$. in other words, as long as $i < k$ there are leaves left at depth $\ell_{\max}$. The last codeword only needs one leaf since it is of maximum length, which shows that it is always possible to construct a prefix-free code if the inequality is fulfilled.

As a complement to Kraft inequality we also give, without proof, the generalization by McMillan for uniquely decodable codes.

Theorem 25 (McMillan) *There exists a **uniquely decodable** D**-ary code** with codeword lengths $\ell_1, \ell_2, \ldots, \ell_k$ if and only if*

$$\sum_{i=1}^{k} D^{-\ell_i} \leq 1$$

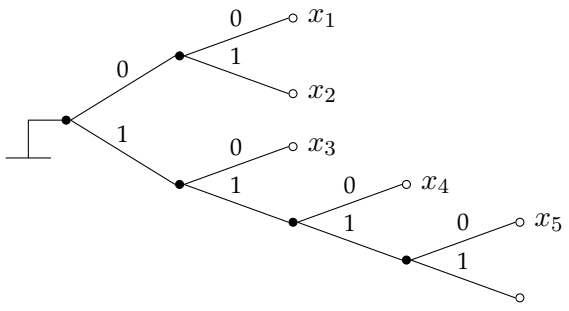
□

Notice that the second part of the proof is constructive. It describes a method to find a tree representing a prefix-free code for the given codeword lengths.

Example 32 To construct a binary prefix-free code with codeword lengths $\ell \in \{2, 2, 2, 3, 4\}$, first check that it is possible according to Kraft inequality. The derivation

$$\sum_{i}^{-\ell_i} = 2^{-2} + 2^{-2} + 2^{-2} + 2^{-3} + 2^{-4} = 3\tfrac{1}{4} + \tfrac{1}{8} + \tfrac{1}{16} = \tfrac{15}{16} < 1$$

shows that there exists such code. For example we can use the following binary tree representation to get the tabular to the right.



x	$\mathcal{C}_1$	ℓ
x_1	00	2
x_2	01	2
x_3	10	2
x_4	110	3
x_5	1110	4

Notice that there is one unused leaf in the tree. In this example it means that the codeword for x_5 is unnecessary long and can be shortened to 111.

In the example we see how we can use the tree representation to get a code.

Example 33 To construct a binary prefix-free code with codeword lengths $\ell \in \{1, 2, 2, 3, 4\}$, first check that it is possible according to Kraft inequality. However, the derivation

$$\sum_{i}^{-\ell_i} = 2^{-1} + 2^{-2} + 2^{-2} + 2^{-3} + 2^{-4} = \tfrac{1}{2} + 2\tfrac{1}{4} + \tfrac{1}{8} + \tfrac{1}{16} = \tfrac{19}{16} > 1$$

shows that it is not possible to construct such code. The leaves in the binary tree will not be enough and it is impossible to fit these lengths in it.

With Kraft inequality we have a mathematical condition that must be fulfilled to construct a prefix-free code. With this it is possible to set up an optimization function for the codeword lengths. One standard method for minimization of a function with some side criterion is the Lagrange multiplication method. For this we set up an optimization function

$$J = \sum_i p(x_i)\ell_i + \lambda\Big(\sum_i D^{-\ell_i} - 1\Big)$$

Taking the derivative of J equal zero for each ℓ_k we get an equation system to solve,

$$\frac{\partial}{\partial \ell_k} J = p(x_k) - \lambda D^{-\ell_k} \ln D = 0$$

which gives

$$D^{-\ell_k} = \frac{p(x_k)}{\lambda \ln D} \tag{5.5}$$

We can use the condition from Kraft inequality to get

$$\sum_i D^{-\ell_k} = \sum_i \frac{p(x_k)}{\lambda \ln D} = \frac{1}{\lambda \ln D} \leq 1 \tag{5.6}$$

Combining (5.5) and (5.6) we get

$$D^{-\ell_k} = p(x_k)\frac{1}{\lambda \ln D} \leq p(x_k)$$

Taking the logarithm and multiplying with (-1) we can obtain the optimal codeword length for codeword k as

$$\ell_k^{(\text{opt})} \geq -\log_D p(x_k) \tag{5.7}$$

The average codeword length for a prefix-free code can now be derived[3]

$$L_{\text{opt}} = \sum_i p(x_i)\ell_i^{(\text{opt})} \geq -\sum_i p(x_i)\log_D p(x_i) = H_D(X) = \frac{H(X)}{\log D}$$

We state this result as the following theorem.

Theorem 26 *The codeword length $L = E[\ell(x)]$ of a prefix-free code is lower bounded by the entropy of the source, i.e.*

$$L \geq H_D(X) = \frac{H(X)}{\log D} \tag{5.8}$$

with equality if and only if $\ell(x) = -\log_D p(x)$. □

[3]The notation

$$H_D(X) = \sum_x p(x)\log_D p(x) = \sum_x p(x)\tfrac{\log p(x)}{\log D} = \frac{H(X)}{\log D}$$

is used for the entropy when derived over the base D instead of base 2.

From the previous theorem on the optimal codeword length and the construction method in proof of Kraft inequality, we can find a method to design a code. The codeword length for source symbol x_i is $\ell_i^{(\text{opt})} \geq -\log_D p(x_i)$. This might not be an integer so we need to take the closest integer that fulfill the inequality, i.e. let

$$\ell_i = \lceil -\log_D p(x_i) \rceil$$

To see that we can actually use this for a code we need first to check that Kraft inequality is fulfilled,

$$\sum_i D^{-\ell_i} = \sum_i D^{-\lceil -\log_D p(x_i) \rceil} \leq \sum_i D^{-(-\log_D p(x_i))} = \sum_i p(x_i) = 1$$

which shows that it is possible to construct such a code. This code construction is named *Shannon-Fano Code*.

For this construction the codeword lengths are in the interval

$$-\log_D p(x_i) \leq \ell_i \leq -\log_D p(x_i) + 1$$

Taking the expectation of the above gives

$$E\big[-\log_D p(x)\big] \leq E\big[\ell\big] \leq E\big[-\log_D p(x) + 1\big]$$

which can also be expressed as

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

Summarizing the above, we can use the Shannon-Fano code construction to get a code with maximum codeword length

$$L < H_D(X) + 1$$

Defining an optimum code as one with a minimum codeword length, this can clearly not exceed the Shannon-Fano codeword length. Together with the result in Theorem 26 we can state the following theorem.

Theorem 27 *Given a stochastic variable X, the average codeword length for an optimal D-ary prefix-free code satisfies*

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

□

The following example show that the Shannon-Fano code is not necessarily an optimal code.

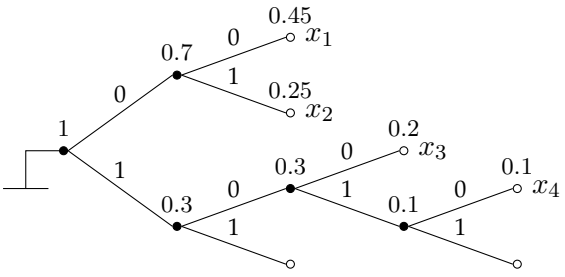
Example 34 Consider a random variable with four outcomes according to the table below. In the table it is also listed the optimal codeword lengths and the lengths for the codewords in a Shannon-Fano code.

x	$p(x)$	$-\log p(x)$	$\ell = \lceil -\log p(cx) \rceil$
x_1	0.45	1.152	2
x_2	0.25	2	2
x_3	0.20	2.32	3
x_4	0.10	3.32	4

From above we know that Kraft inequality is fulfilled, but as a further clarification we derive it

$$\sum_{\ell} 2^{-\ell} = \tfrac{1}{4} + \tfrac{1}{4} + \tfrac{1}{8} + \tfrac{1}{16} = \tfrac{11}{16} < 1$$

This shows that it is possible to construct a prefix-free code with the listed codeword lengths. Following the procedure described earlier we get the following binary tree representation and code list.



x	$p(x)$	$\mathcal{C}$
x_1	0.45	00
x_2	0.25	01
x_3	0.20	100
x_4	0.10	1010

In the tree following the paths for 11 and 1011, there are unused leaves. The first one can be used to show that this code is not optimal. By moving the label for x_4 to the leaf at 11 the length for this codeword will decrease from 4 to 2, which will result in a code with lower average codeword length. Hence, the Shanon-Fano construction does not in general give an optimal code. However, it should be bounded by (5.8). To view this we first derive the entropy for the random variable

$$H(X) = H(0.45, 0.25, 0.2, 0.1) = 1.815$$

With use of the path length lemma we can derive the average codeword length

$$L = 1 + 0.7 + 0.3 + 0.3 + 0.1 = 2.4$$

which lies in between $H(X) = 1.815$ and $H(X) + 1 = 2.815$.

To see how this relates to random processes we view sequences $\boldsymbol{x} = x_1 x_2 \dots x_n$ of length n as the alphabet of the source. Then we are interested in the average codeword length per source symbol in the vector. By using Theorem 26 it is possible to bound the optimal codeword length as in the next corollary

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 $X_1X_2 \dots X_n$ 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 Coding

To be done.