Chapter 2

Coding for Discrete Sources

2.1 Introduction

A general block diagram of a point-to-point digital communication system was given in Figure 1.1. The source encoder converts the sequence of symbols from the source to a sequence of binary digits, preferably using as few binary digits per symbol as possible. The source decoder performs the inverse operation. Initially, in the spirit of source/channel separation, we ignore the possibility that errors are made in the channel decoder and assume that the source decoder operates on the source encoder output.

We first distinguish between three important classes of sources:

- **Discrete sources**

  The output of a discrete source is a sequence of symbols from a known discrete alphabet $\mathcal{X}$. This alphabet could be the alphanumeric characters, the characters on a computer keyboard, English letters, Chinese characters, the symbols in sheet music (arranged in some systematic fashion), binary digits, etc.

  The discrete alphabets in this chapter are assumed to contain a finite set of symbols.[1]

  It is often convenient to view the sequence of symbols as occurring at some fixed rate in time, but there is no need to bring time into the picture (for example, the source sequence might reside in a computer file and the encoding can be done off-line).

  This chapter focuses on source coding and decoding for discrete sources." Supplementary references for source coding are Chapter 3 of [7] and Chapter 5 of [4]. A more elementary partial treatment is in Sections 4.1-4.3 of [22].

- **Analog waveform sources**

  The output of an analog source, in the simplest case, is an analog real waveform, representing, for example, a speech waveform. The word analog is used to emphasize that the waveform can be arbitrary and is not restricted to taking on amplitudes from some discrete set of values.

[1]A set is usually defined to be discrete if it includes either a finite or countably infinite number of members. The countably infinite case does not extend the basic theory of source coding in any important way, but it is occasionally useful in looking at limiting cases, which will be discussed as they arise.

It is also useful to consider analog waveform sources with outputs that are complex functions of time; both real and complex waveform sources are discussed later.

More generally, the output of an analog source might be an image (represented as an intensity function of horizontal/vertical location) or video (represented as an intensity function of horizontal/vertical location and time). For simplicity, we restrict our attention to analog waveforms, mapping a single real variable, time, into a real or complex-valued intensity.

- **Discrete-time sources with analog values (analog sequence sources)**

  These sources are halfway between discrete and analog sources. The source output is a sequence of real numbers (or perhaps complex numbers). Encoding such a source is of interest in its own right, but is of interest primarily as a subproblem in encoding analog sources. That is, analog waveform sources are almost invariably encoded by first either sampling the analog waveform or representing it by the coefficients in a series expansion. Either way, the result is a sequence of numbers, which is then encoded.

There are many differences between discrete sources and the latter two types of analog sources. The most important is that a discrete source can be, and almost always is, encoded in such a way that the source output can be uniquely retrieved from the encoded string of binary digits. Such codes are called *uniquely decodable*[2]. On the other hand, for analog sources, there is usually no way to map the source values to a bit sequence such that the source values are uniquely decodable. For example, an infinite number of binary digits is required for the exact specification of an arbitrary real number between 0 and 1. Thus, some sort of quantization is necessary for these analog values, and this introduces distortion. Source encoding for analog sources thus involves a trade-off between the bit rate and the amount of distortion.

Analog sequence sources are almost invariably encoded by first quantizing each element of the sequence (or more generally each successive n-tuple of sequence elements) into one of a finite set of symbols. This symbol sequence is a discrete sequence which can then be encoded into a binary sequence.

Figure 2.1 summarizes this layered view of analog and discrete source coding. As illustrated, discrete source coding is both an important subject in its own right for encoding text-like sources, but is also the inner layer in the encoding of analog sequences and waveforms.

The remainder of this chapter discusses source coding for discrete sources. The following chapter treats source coding for analog sequences and the fourth chapter treats waveform sources.

2.2 Fixed-length codes for discrete sources

The simplest approach to encoding a discrete source into binary digits is to create a code $\mathcal{C}$ that maps each symbol x of the alphabet $\mathcal{X}$ into a distinct codeword $\mathcal{C}(x)$, where $\mathcal{C}(x)$ is a block of binary digits. Each such block is restricted to have the same block length L, which is why the code is called a *fixed-length code.*

[2]Uniquely-decodable codes are sometimes called noiseless codes in elementary treatments. *Uniquely decodable* captures both the intuition and the precise meaning far better than *noiseless*. Unique decodability is defined shortly.

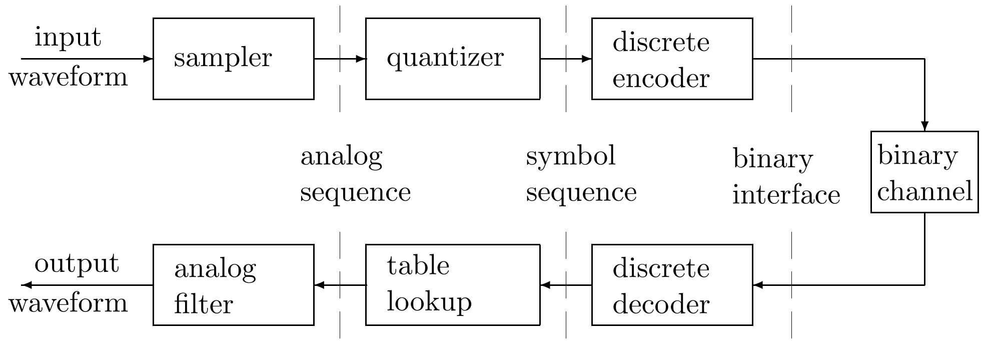


Figure 2.1: Discrete sources require only the inner layer above, whereas the inner two layers are used for analog sequences and all three layers are used for waveforms sources.

For example, if the alphabet $\mathcal{X}$ consists of the 7 symbols $\{a, b, c, d, e, f, g\}$, then the following fixed-length code of block length $L = 3$ could be used.

$$\begin{aligned}
\mathcal{C}(a) &= \ 000 \\
\mathcal{C}(b) &= \ 001 \\
\mathcal{C}(c) &= \ 010 \\
\mathcal{C}(d) &= \ 011 \\
\mathcal{C}(e) &= \ 100 \\
\mathcal{C}(f) &= \ 101 \\
\mathcal{C}(g) &= \ 110.
\end{aligned}$$

The source output, $x_1, x_2, \dots,$ would then be encoded into the encoded output $\mathcal{C}(x_1)\mathcal{C}(x_2)\dots$ and thus the encoded output contains L bits per source symbol. For the above example the source sequence $bad\dots$ would be encoded into $001000011\dots.$ Note that the output bits are simply run together (or, more technically, concatenated).

There are 2^L different combinations of values for a block of L bits. Thus, if the number of symbols in the source alphabet, $M = |\mathcal{X}|$, satisfies $M \le 2^L$, then a different binary L-tuple may be assigned to each symbol. Assuming that the decoder knows where the beginning of the encoded sequence is, the decoder can segment the sequence into L bit blocks and then decode each block into the corresponding source symbol.

In summary, if the source alphabet has size M, then this coding method requires $L = \lceil \log_2 M \rceil$ bits to encode each source symbol, where $\lceil w \rceil$ denotes the smallest integer greater than or equal to the real number w. Thus $\log_2 M \le L < \log_2 M + 1$. The lower bound, $\log_2 M$, can be achieved with equality if and only if M is a power of 2.

A technique to be used repeatedly is that of first segmenting the sequence of source symbols into successive blocks of n source symbols at a time. Given an alphabet $\mathcal{X}$ of M symbols, there are M^n possible n-tuples. These M^n n-tuples are regarded as the elements of a super-alphabet. Each n-tuple can be encoded rather than encoding the original symbols. Using fixed-length source coding on these n-tuples, each source n-tuple can be encoded into $L = \lceil \log_2 M^n \rceil$ bits.

The rate $\overline{L} = L/n$ of encoded bits per original source symbol is then bounded by

$$\begin{aligned} \overline{L} = \frac{\lceil \log_2 M^n \rceil}{n} &\geq \frac{n \log_2 M}{n} = \log_2 M; \\ \overline{L} = \frac{\lceil \log_2 M^n \rceil}{n} &< \frac{n(\log_2 M) + 1}{n} = \log_2 M + \frac{1}{n}. \end{aligned}$$

Thus $\log_2 M \leq \overline{L} < \log_2 M + \frac{1}{n}$, and by letting n become sufficiently large, the average number of coded bits per source symbol can be made arbitrarily close to $\log_2 M$, regardless of whether M is a power of 2.

Some remarks:

- This simple scheme to make $\overline{L}$ arbitrarily close to $\log_2 M$ is of greater theoretical interest than practical interest. As shown later, $\log_2 M$ is the minimum possible binary rate for uniquely-decodable source coding if the source symbols are independent and equiprobable. Thus this scheme asymptotically approaches this minimum.

- This result begins to hint at why measures of information are logarithmic in the alphabet size.[3] The logarithm is usually taken to the base 2 in discussions of binary codes. Henceforth $\log n$ means "$\log_2 n$."

- This method is nonprobabilistic; it takes no account of whether some symbols occur more frequently than others, and it works robustly regardless of the symbol frequencies. But if it is known that some symbols occur more frequently than others, then the rate $\overline{L}$ of coded bits per source symbol can be reduced by assigning shorter bit sequences to more common symbols in a *variable-length source code*. This will be our next topic.

2.3 Variable-length codes for discrete sources

The motivation for using variable-length encoding on discrete sources is the intuition that data compression can be achieved by mapping more probable symbols into shorter bit sequences, and less likely symbols into longer bit sequences. This intuition was used in the Morse code of old-time telegraphy in which letters were mapped into strings of dots and dashes, using shorter strings for common letters and longer strings for less common letters.

A *variable-length code* $\mathcal{C}$ maps each source symbol a_j in a source alphabet $\mathcal{X} = \{a_1, \dots, a_M\}$ to a binary string $\mathcal{C}(a_j)$, called a *codeword*. The number of bits in $\mathcal{C}(a_j)$ is called the *length* $l(a_j)$ of $\mathcal{C}(a_j)$. For example, a variable-length code for the alphabet $\mathcal{X} = \{a, b, c\}$ and its lengths might be given by

$$\begin{array}{lll} \mathcal{C}(a) = & 0 & \qquad l(a) = 1 \\ \mathcal{C}(b) = & 10 & \qquad l(b) = 2 \\ \mathcal{C}(c) = & 11 & \qquad l(c) = 2 \end{array}$$

Successive codewords of a variable-length code are assumed to be transmitted as a continuing sequence of bits, with no demarcations of codeword boundaries (*i.e.*, no commas or spaces). The

[3]The notion that information can be viewed as a logarithm of a number of possibilities was first suggested by Hartley [11] in 1927.

source decoder, given an original starting point, must determine where the codeword boundaries are; this is called *parsing.*

A potential system issue with variable-length coding is the requirement for buffering. If source symbols arrive at a fixed rate and the encoded bit sequence must be transmitted at a fixed bit rate, then a buffer must be provided between input and output. This requires some sort of recognizable 'fill' to be transmitted when the buffer is empty and the possibility of lost data when the buffer is full. There are many similar system issues, including occasional errors on the channel, initial synchronization, terminal synchronization, etc. Many of these issues are discussed later, but they are more easily understood after the more fundamental issues are discussed.

2.3.1 Unique decodability

The major property that is usually required from any variable-length code is that of *unique decodability.* This essentially means that for any sequence of source symbols, that sequence can be reconstructed unambiguously from the encoded bit sequence. Here initial synchronization is assumed: the source decoder knows which is the first bit in the coded bit sequence. Note that without initial synchronization, not even a fixed-length code can be uniquely decoded.

Clearly, unique decodability requires that $\mathcal{C}(a_j) \neq \mathcal{C}(a_i)$ for each $i \neq j$. More than that, however, it requires that strings[4] of encoded symbols be distinguishable. The following definition says this precisely:

Definition 2.3.1. A code $\mathcal{C}$ for a discrete source is uniquely decodable if, for any string of source symbols, say $x_1, x_2, \dots, x_n$, the concatenation[5] of the corresponding codewords, $\mathcal{C}(x_1)\mathcal{C}(x_2)\cdots\mathcal{C}(x_n)$, differs from the concatenation of the codewords $\mathcal{C}(x'_1)\mathcal{C}(x'_2)\cdots\mathcal{C}(x'_m)$ for any other string $x'_1, x'_2, \dots, x'_m$ of source symbols.

In other words, $\mathcal{C}$ is uniquely decodable if all concatenations of codewords are distinct.

Remember that there are no commas or spaces between codewords; the source decoder has to determine the codeword boundaries from the received sequence of bits. (If commas were inserted, the code would be ternary rather than binary.)

For example, the above code $\mathcal{C}$ for the alphabet $\mathcal{X} = \{a, b, c\}$ is soon shown to be uniquely decodable. However, the code $\mathcal{C}'$ defined by

$$\begin{aligned} \mathcal{C}'(a) &= \ 0 \\ \mathcal{C}'(b) &= \ 1 \\ \mathcal{C}'(c) &= \ 01 \end{aligned}$$

is not uniquely decodable, even though the codewords are all different. If the source decoder observes 01, it cannot determine whether the source emitted $(a\, b)$ or (c).

Note that the property of unique decodability depends only on the set of codewords and not on the mapping from symbols to codewords. Thus we can refer interchangeably to uniquely-decodable codes and uniquely-decodable codeword sets.

[4]A *string* of symbols is an n-tuple of symbols for any finite n. A *sequence* of symbols is an n-tuple in the limit $n \to \infty$, although the word sequence is also used when the length might be either finite or infinite.

[5]The concatenation of two strings, say $u_1 \cdots u_l$ and $v_1 \cdots v_{l'}$ is the combined string $u_1 \cdots u_l v_1 \cdots v_{l'}$.

2.3.2 Prefix-free codes for discrete sources

Decoding the output from a uniquely-decodable code, and even determining whether it is uniquely decodable, can be quite complicated. However, there is a simple class of uniquely-decodable codes called *prefix-free codes*. As shown later, these have the following advantages over other uniquely-decodable codes:[6]

- If a uniquely-decodable code exists with a certain set of codeword lengths, then a prefix-free code can easily be constructed with the same set of lengths.
- The decoder can decode each codeword of a prefix-free code immediately on the arrival of the last bit in that codeword.
- Given a probability distribution on the source symbols, it is easy to construct a prefix-free code of minimum expected length.

Definition 2.3.2. A *prefix* of a string $y_1 \cdots y_l$ is any initial substring $y_1 \cdots y_{l'}$, $l' \le l$ of that string. The prefix is *proper* if $l' < l$. A code is *prefix-free* if no codeword is a prefix of any other codeword.

For example, the code $\mathcal{C}$ with codewords $0, 10$, and 11 is prefix-free, but the code $\mathcal{C}'$ with codewords 0, 1, and 01 is not. Every fixed-length code with distinct codewords is prefix-free.

We will now show that every prefix-free code is uniquely decodable. The proof is constructive, and shows how the decoder can uniquely determine the codeword boundaries.

Given a prefix-free code $\mathcal{C}$, a corresponding *binary code tree* can be defined which grows from a root on the left to leaves on the right representing codewords. Each branch is labelled 0 or 1 and each node represents the binary string corresponding to the branch labels from the root to that node. The tree is extended just enough to include each codeword. That is, each node in the tree is either a codeword or proper prefix of a codeword (see Figure 2.2).

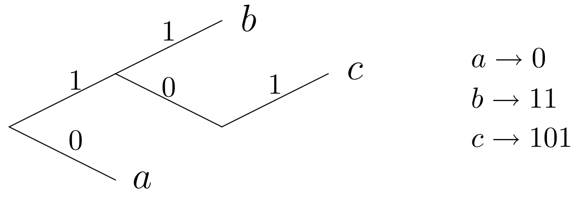
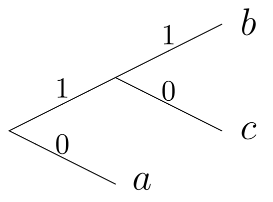


Figure 2.2: The binary code tree for a prefix-free code.

The prefix-free condition ensures that each codeword corresponds to a leaf node (*i.e.*, a node with no adjoining branches going to the right). Each intermediate node (*i.e.*, nodes having one or more adjoining branches going to the right) is a prefix of some codeword reached by traveling right from the intermediate node.

[6]With all the advantages of prefix-free codes, it is difficult to understand why the more general class is even discussed. This will become clearer much later.

The tree of Figure 2.2 has an intermediate node, 10, with only one right-going branch. This shows that the codeword for c could be shortened to 10 without destroying the prefix-free property. This is shown in Figure 2.3.

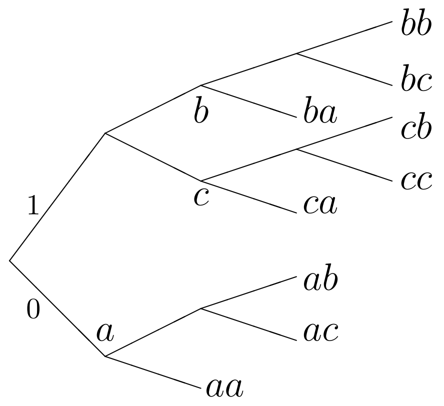


$a \to 0$
$b \to 11$
$c \to 10$

Figure 2.3: A code with shorter lengths than that of Figure 2.2.

A prefix-free code will be called *full* if no new codeword can be added without destroying the prefix-free property. As just seen, a prefix-free code is also full if no codeword can be shortened without destroying the prefix-free property. Thus the code of Figure 2.2 is not full, but that of Figure 2.3 is.

To see why the prefix-free condition guarantees unique decodability, consider the tree for the concatenation of two codewords. This is illustrated in Figure 2.4 for the code of Figure 2.3. This new tree has been formed simply by grafting a copy of the original tree onto each of the leaves of the original tree. Each concatenation of two codewords thus lies on a different node of the tree and also differs from each single codeword. One can imagine grafting further trees onto the leaves of Figure 2.4 to obtain a tree representing still more codewords concatenated together. Again all concatenations of code words lie on distinct nodes, and thus correspond to distinct binary strings.



$aa \to 00$
$ab \to 011$
$ac \to 010$
$ba \to 110$
$bb \to 1111$
$bc \to 1110$
$ca \to 100$
$cb \to 1011$
$cc \to 1010$

Figure 2.4: Binary code tree for two codewords; upward branches represent 1's.

An alternative way to see that prefix-free codes are uniquely decodable is to look at the codeword parsing problem from the viewpoint of the source decoder. Given the encoded binary string for any strong of source symbols, the source decoder can decode the first symbol simply by reading the string from left to right and following the corresponding path in the code tree until it reaches a leaf, which must correspond to the first codeword by the prefix-free property. After stripping off the first codeword, the remaining binary string is again a string of codewords, so the source decoder can find the second codeword in the same way, and so on *ad infinitum*.

For example, suppose a source decoder for the code of Figure 2.3 decodes the sequence $1010011\cdots$. Proceeding through the tree from the left, it finds that 1 is not a codeword, but that 10 is the codeword for c. Thus c is decoded as the first symbol of the source output, leaving the string $10011\cdots$. Then c is decoded as the next symbol, leaving $011\cdots$, which is decoded into a and then b, and so forth.

This proof also shows that prefix-free codes can be decoded with no delay. As soon as the final bit of a codeword is received at the decoder, the codeword can be recognized and decoded without waiting for additional bits. For this reason, prefix-free codes are sometimes called instantaneous codes.

It has been shown that all prefix-free codes are uniquely decodable. The converse is not true, as shown by the following code:

$$\begin{aligned} \mathcal{C}(a) &= \ 0 \\ \mathcal{C}(b) &= \ 01 \\ \mathcal{C}(c) &= \ 011 \end{aligned}$$

An encoded sequence for this code can be uniquely parsed by recognizing 0 as the beginning of each new code word. A different type of example is given in Exercise 2.6.

With variable-length codes, if there are errors in data transmission, then the source decoder may lose codeword boundary synchronization and may make more than one symbol error. It is therefore important to study the synchronization properties of variable-length codes. For example, the prefix-free code $\{0, 10, 110, 1110, 11110\}$ is instantaneously self-synchronizing, because every 0 occurs at the end of a codeword. The shorter prefix-free code $\{0, 10, 110, 1110, 1111\}$ is probabilistically self-synchronizing; again, any observed 0 occurs at the end of a codeword, but since there may be a sequence of 1111 codewords of unlimited length, the length of time before resynchronization is a random variable. These questions are not pursued further here.

2.3.3 The Kraft inequality for prefix-free codes

The Kraft inequality [17] is a condition determining whether it is possible to construct a prefix-free code for a given discrete source alphabet $\mathcal{X} = \{a_1, \dots, a_M\}$ with a given set of codeword lengths $\{l(a_j); 1 \le j \le M\}$.

Theorem 2.3.1 (Kraft inequality for prefix-free codes). *Every prefix-free code for an alphabet $\mathcal{X} = \{a_1, \dots, a_M\}$ with codeword lengths $\{l(a_j); 1 \le j \le M\}$ satisfies*

$$\sum_{j=1}^{M} 2^{-l(a_j)} \le 1. \tag{2.1}$$

Conversely, if (2.1) is satisfied, then a prefix-free code with lengths $\{l(a_j); 1 \le j \le M\}$ exists.

Moreover, every full prefix-free code satisfies (2.1) with equality and every non-full prefix-free code satisfies it with strict inequality.

For example, this theorem implies that there exists a full prefix-free code with codeword lengths $\{1, 2, 2\}$ (two such examples have already been given), but there exists no prefix-free code with codeword lengths $\{1, 1, 2\}$.

Before proving the theorem, we show how to represent codewords as base 2 expansions (the base 2 analog of base 10 decimals) in the binary number system. After understanding this representation, the theorem will be almost obvious. The base 2 expansion $.y_1y_2\cdots y_l$ represents the rational number $\sum_{m=1}^{l} y_m 2^{-m}$. For example, .011 represents $1/4 + 1/8$.

Ordinary decimals with l digits are frequently used to indicate an approximation of a real number to l places of accuracy. Here, in the same way, the base 2 expansion $.y_1y_2\cdots y_l$ is viewed as 'covering' the interval[7] $[\sum_{m=1}^{l} y_m 2^{-m},\ \sum_{m=1}^{l} y_m 2^{-m} + 2^{-l})$. This interval has size 2^{-l} and includes all numbers whose base 2 expansions start with $.y_1 \dots y_l$.

In this way, any codeword $\mathcal{C}(a_j)$ of length l is represented by a rational number in the interval [0, 1) and covers an interval of size 2^{-l} which includes all strings that contain $\mathcal{C}(a_j)$ as a prefix (see Figure 2.3). The proof of the theorem follows:

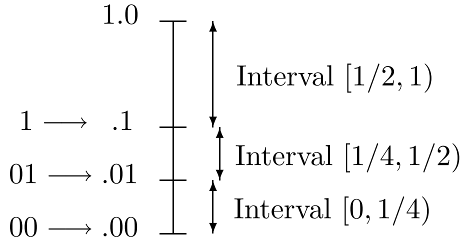


Figure 2.5: Base 2 expansion numbers and intervals representing codewords. The codewords represented above are (00, 01, and 1).

Proof: First, assume that $\mathcal{C}$ is a prefix-free code with codeword lengths $\{l(a_j),\ 1 \le j \le M\}$. For any distinct a_j and a_i in $\mathcal{X}$, it was shown above that the base 2 expansion corresponding to $\mathcal{C}(a_j)$ cannot lie in the interval corresponding to $\mathcal{C}(a_i)$ since $\mathcal{C}(a_i)$ is not a prefix of $\mathcal{C}(a_j)$. Thus the lower end of the interval corresponding to any codeword $\mathcal{C}(a_j)$ cannot lie in the interval corresponding to any other codeword. Now, if two of these intervals intersect, then the lower end of one of them must lie in the other, which is impossible. Thus the two intervals must be disjoint and thus the set of all intervals associated with the codewords are disjoint. Since all these intervals are contained in the interval $[0, 1)$ and the size of the interval corresponding to $\mathcal{C}(a_j)$ is $2^{-l(a_j)}$, (2.1) is established.

Next note that if (2.1) is satisfied with strict inequality, then some interval exists in $[0, 1)$ that does not intersect any codeword interval; thus another codeword can be 'placed' in this interval and the code is not full. If (2.1) is satisfied with equality, then the intervals fill up $[0, 1)$. In this case no additional code word can be added and the code is full.

Finally we show that a prefix-free code can be constructed from any desired set of codeword lengths $\{l(a_j),\ 1 \le j \le M\}$ for which (2.1) is satisfied. Put the set of lengths in nondecreasing order, $l_1 \le l_2 \le \cdots \le l_M$ and let $u_1, \dots, u_M$ be the real numbers corresponding to the codewords in the construction to be described. The construction is quite simple: $u_1 = 0$, and for all

[7]Brackets and parentheses, respectively, are used to indicate closed and open boundaries; thus the interval $[a, b)$ means the set of real numbers u such that $a \le u < b$.

$j, 1 < j \leq M$,

$$u_j = \sum_{i=1}^{j-1} 2^{-l_i}. \tag{2.2}$$

Each term on the right is an integer multiple of 2^{-l_j}, so u_j is also an integer multiple of 2^{-l_j}. From (2.1), $u_j < 1$, so u_j can be represented by a base 2 expansion with l_j places. The corresponding codeword of length l_j can be added to the code while preserving prefix-freedom (see Figure 2.6).

□

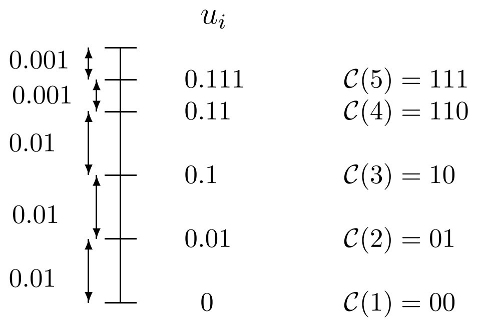


Figure 2.6: Construction of codewords for the set of lengths $\{2, 2, 2, 3, 3\}$. $\mathcal{C}(i)$ is formed from u_i by representing u_i to l_i places.

Some final remarks on the Kraft inequality:

- Just because a code has lengths that satisfy (2.1), it does not follow that the code is prefix-free, or even uniquely decodable.
- Exercise 2.11 shows that Theorem 2.3.1 also holds for all uniquely-decodable codes— *i.e.*, there exists a uniquely-decodable code with codeword lengths $\{l(a_j), 1 \leq j \leq M\}$ if and only if (2.1) holds. This will imply that if a uniquely-decodable code exists with a certain set of codeword lengths, then a prefix-free code exists with the same set of lengths. So why use any code other than a prefix-free code?

2.4 Probability models for discrete sources

It was shown above that prefix-free codes exist for any set of codeword lengths satisfying the Kraft inequality. When does it desirable to use one of these codes?– *i.e.*, when is the expected number of coded bits per source symbol less than $\log M$ and why is the expected number of coded bits per source symbol the primary parameter of importance?

This question cannot be answered without a probabilistic model for the source. For example, the $M = 4$ prefix-free set of codewords $\{0, 10, 110, 111\}$ has an expected length of $2.25 > 2 = \log M$ if the source symbols are equiprobable, but if the source symbol probabilities are $\{1/2, 1/4, 1/8, 1/8\}$, then the expected length is $1.75 < 2$.

The discrete sources that one meets in applications usually have very complex statistics. For example, consider trying to compress email messages. In typical English text, some letters such as e and o occur far more frequently than q, x, and z. Moreover, the letters are not independent; for example h is often preceded by t, and q is almost always followed by u. Next, some strings of letters are words, while others are not; those that are not have probability near 0 (if in fact the text is correct English). Over longer intervals, English has grammatical and semantic constraints, and over still longer intervals, such as over multiple email messages, there are still further constraints.

It should be clear therefore that trying to find an accurate probabilistic model of a real-world discrete source is not going to be a productive use of our time. An alternative approach, which has turned out to be very productive, is to start out by trying to understand the encoding of "toy" sources with very simple probabilistic models. After studying such toy sources, it will be shown how to generalize to source models with more and more general structure, until, presto, real sources can be largely understood even without good stochastic models. This is a good example of a problem where having the patience to look carefully at simple and perhaps unrealistic models pays off handsomely in the end.

The type of toy source that will now be analyzed in some detail is called a discrete memoryless source.

2.4.1 Discrete memoryless sources

A *discrete memoryless source* (DMS) is defined by the following properties:

- The source output is an unending sequence, $X_1, X_2, X_3, \ldots$, of randomly selected symbols from a *finite* set $\mathcal{X} = \{a_1, a_2, \ldots, a_M\}$, called the *source alphabet*.
- Each source output $X_1, X_2, \ldots$ is selected from $\mathcal{X}$ using the same probability mass function (pmf) $\{p_X(a_1), \ldots, p_X(a_M)\}$. Assume that $p_X(a_j) > 0$ for all j, $1 \le j \le M$, since there is no reason to assign a code word to a symbol of zero probability and no reason to model a discrete source as containing impossible symbols.
- Each source output X_k is statistically independent of the previous outputs $X_1, \ldots, X_{k-1}$.

The randomly chosen symbols coming out of the source are called *random symbols*. They are very much like random variables except that they may take on nonnumeric values. Thus, if X denotes the result of a fair coin toss, then it can be modeled as a random symbol that takes values in the set {HEADS, TAILS} with equal probability. Note that if X is a nonnumeric random symbol, then it makes no sense to talk about its expected value. However, the notion of statistical independence between random symbols is the same as that for random variables, *i.e.*, the event that X_i is any given element of $\mathcal{X}$ is independent of the events corresponding to the values of the other random symbols.

The word memoryless in the definition refers to the statistical independence between different random symbols, *i.e.*, each variable is chosen with no memory of how the previous random symbols were chosen. In other words, the source symbol sequence is independent and identically distributed (iid).[8]

In summary, a DMS is a semi-infinite iid sequence of random symbols

$$X_1, X_2, X_3, \ldots$$

[8]Do not confuse this notion of memorylessness with any non-probabalistic notion in system theory.

each drawn from the finite set $\mathcal{X}$, each element of which has positive probability.

A sequence of independent tosses of a biased coin is one example of a DMS. The sequence of symbols drawn (with replacement) in a Scrabble™ game is another. The reason for studying these sources is that they provide the tools for studying more realistic sources.

2.5 Minimum $\overline{L}$ for prefix-free codes

The Kraft inequality determines which sets of codeword lengths are possible for prefix-free codes. Given a discrete memoryless source (DMS), we want to determine what set of codeword lengths can be used to *minimize* the expected length of a prefix-free code for that DMS. That is, we want to minimize the expected length subject to the Kraft inequality.

Suppose a set of lengths $l(a_1), \dots, l(a_M)$ (subject to the Kraft inequality) is chosen for encoding each symbol into a prefix-free codeword. Define $L(X)$ (or more briefly L) as a random variable representing the codeword length for the randomly selected source symbol. The expected value of L for the given code is then given by

$$\overline{L} = \mathsf{E}[L] = \sum_{j=1}^{M} l(a_j) p_X(a_j).$$

We want to find $\overline{L}_{\min}$, which is defined as the minimum value of $\overline{L}$ over all sets of codeword lengths satisfying the Kraft inequality.

Before finding $\overline{L}_{\min}$, we explain why this quantity is of interest. The number of bits resulting from using the above code to encode a long block $\boldsymbol{X} = (X_1, X_2, \dots, X_n)$ of symbols is $S_n = L(X_1) + L(X_2) + \cdots + L(X_n)$. This is a sum of n iid random variables (rv's), and the law of large numbers, which is discussed in Section 2.7.1, implies that S_n/n, the number of bits per symbol in this long block, is very close to $\overline{L}$ with probability very close to 1. In other words, $\overline{L}$ is essentially the rate (in bits per source symbol) at which bits come out of the source encoder. This motivates the objective of finding $\overline{L}_{\min}$ and later of finding codes that achieve the minimum.

Before proceeding further, we simplify our notation. We have been carrying along a completely arbitrary finite alphabet $\mathcal{X} = \{a_1, \dots, a_M\}$ of size $M = |\mathcal{X}|$, but this problem (along with most source coding problems) involves only the probabilities of the M symbols and not their names. Thus define the source alphabet to be $\{1, 2, \dots, M\}$, denote the symbol probabilities by $p_1, \dots, p_M$, and denote the corresponding codeword lengths by $l_1, \dots, l_M$. The expected length of a code is then

$$\overline{L} = \sum_{j=1}^{M} l_j p_j$$

Mathematically, the problem of finding $\overline{L}_{\min}$ is that of minimizing $\overline{L}$ over all sets of integer lengths $l_1, \dots, l_M$ subject to the Kraft inequality:

$$\overline{L}_{\min} = \min_{l_1, \dots, l_M : \sum_j 2^{-l_j} \le 1} \left\{ \sum_{j=1}^{M} p_j l_j \right\}. \qquad (2.3)$$

2.5.1 Lagrange multiplier solution for the minimum $\overline{L}$

The minimization in (2.3) is over a function of M variables, $l_1, \dots, l_M$, subject to constraints on those variables. Initially, consider a simpler problem where there are no integer constraint on the l_j. This simpler problem is then to minimize $\sum_j p_j l_j$ over all real values of $l_1, \dots, l_M$ subject to $\sum_j 2^{-l_j} \leq 1$. The resulting minimum is called $\overline{L}_{\min}$(noninteger).

Since the allowed values for the lengths in this minimization include integer lengths, it is clear that $\overline{L}_{\min}(\text{noninteger}) \leq \overline{L}_{\min}$. This noninteger minimization will provide a number of important insights about the problem, so its usefulness extends beyond just providing a lower bound on $\overline{L}_{\min}$.

Note first that the minimum of $\sum_j l_j p_j$ subject to $\sum_j 2^{-l_j} \leq 1$ must occur when the constraint is satisfied with equality, for otherwise, one of the l_j could be reduced, thus reducing $\sum_j p_j l_j$ without violating the constraint. Thus the problem is to minimize $\sum_j p_j l_j$ subject to $\sum_j 2^{-l_j} = 1$.

Problems of this type are often solved by using a Lagrange multiplier. The idea is to replace the minimization of one function, subject to a constraint on another function, by the minimization of a linear combination of the two functions, in this case the minimization of

$$\sum_j p_j l_j + \lambda \sum_j 2^{-l_j}. \tag{2.4}$$

If the method works, the expression can be minimized for each choice of λ (called a *Lagrange multiplier*); λ can then be chosen so that the optimizing choice of $l_1, \dots, l_M$ satisfies the constraint. The minimizing value of (2.4) is then $\sum_j p_j l_j + \lambda$. This choice of $l_1, \dots, l_M$ minimizes the original constrained optimization, since for any $l'_1, \dots, l'_M$ that satisfies the constraint $\sum_j 2^{-l'_j} = 1$, the expression in (2.4) is $\sum_j p_j l'_j + \lambda$, which must be greater than or equal to $\sum_j p_j l_j + \lambda$.

We can attempt[9] to minimize (2.4) simply by setting the derivitive with respect to each l_j equal to 0. This yields

$$p_j - \lambda(\ln 2)2^{-l_j} = 0; \quad 1 \leq j \leq M. \tag{2.5}$$

Thus $2^{-l_j} = p_j/(\lambda \ln 2)$. Since $\sum_j p_j = 1$, λ must be equal to $1/\ln 2$ in order to satisfy the constraint $\sum_j 2^{-l_j} = 1$. Then $2^{-l_j} = p_j$, or equivalently $l_j = -\log p_j$. It will be shown shortly that this stationary point actually achieves a minimum. Substituting this solution into (2.3),

$$\overline{L}_{\min}(\text{noninteger}) = -\sum_{j=1}^{M} p_j \log p_j. \tag{2.6}$$

The quantity on the right side of (2.6) is called the *entropy*[10] of X, and denoted as $\mathsf{H}[X]$. Thus

$$\mathsf{H}[X] = -\sum_j p_j \log p_j.$$

[9]There are well-known rules for when the Lagrange multiplier method works and when it can be solved simply by finding a stationary point. The present problem is so simple, however, that this machinery is unnecessary.

[10]Note that X is a random symbol and carries with it all of the accompanying baggage, including a pmf. The entropy $\mathsf{H}[X]$ is a numerical function of the random symbol including that pmf; in the same way $\mathsf{E}[L]$ is a numerical function of the rv L. Both $\mathsf{H}[X]$ and $\mathsf{E}[L]$ are expected values of particular rv's. In distinction, $L(X)$ above is an rv in its own right; it is based on some function $l(x)$ mapping $\mathcal{X} \to \mathbb{R}$ and takes the sample value $l(x)$ for all sample points such that $X = x$.

In summary, the entropy $\mathsf{H}[X]$ is a lower bound to $\overline{L}$ for prefix-free codes and this lower bound is achieved when $l_j = -\log p_j$ for each j. The bound was derived by ignoring the integer constraint, and can be met only if $-\log p_j$ is an integer for each j; *i.e.*, if each p_j is a power of 2.

2.5.2 Entropy bounds on $\overline{L}$

We now return to the problem of minimizing $\overline{L}$ with an integer constraint on lengths. The following theorem both establishes the correctness of the previous non-integer optimization and provides an upper bound on $\overline{L}_{\min}$.

Theorem 2.5.1 (Entropy bounds for prefix-free codes). *Let X be a discrete random symbol with symbol probabilities $p_1, \ldots, p_M$. Let $\overline{L}_{\min}$ be the minimum expected codeword length over all prefix-free codes for X. Then*

$$H[X] \le \overline{L}_{\min} < H[X] + 1 \quad \text{bit/symbol.} \tag{2.7}$$

Furthermore, $\overline{L}_{\min} = H[X]$ if and only if each probability p_j is an integer power of 2.

Proof: It is first shown that $\mathsf{H}[X] \le \overline{L}$ for all prefix-free codes. Let $l_1, \ldots, l_M$ be the codeword lengths of an arbitrary prefix-free code. Then

$$\mathsf{H}[X] - \overline{L} \;=\; \sum_{j=1}^{M} p_j \log \frac{1}{p_j} - \sum_{j=1}^{M} p_j l_j \;=\; \sum_{j=1}^{M} p_j \log \frac{2^{-l_j}}{p_j}, \tag{2.8}$$

where $\log 2^{-l_j}$ has been substituted for $-l_j$.

We now use the very useful inequality $\ln u \le u - 1$, or equivalently $\log u \le (\log e)(u-1)$, which is illustrated in Figure 2.7. Note that equality holds only at the point $u = 1$.

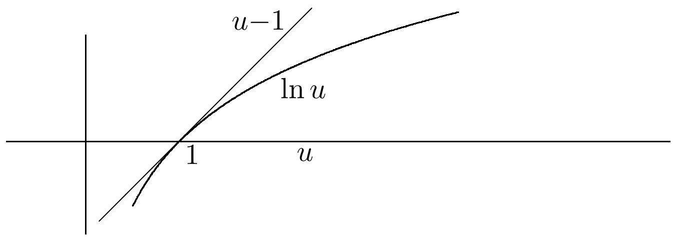


Figure 2.7: The inequality $\ln u \le u - 1$. The inequality is strict except at $u = 1$.

Substituting this inequality in (2.8),

$$\mathsf{H}[X] - \overline{L} \;\le\; (\log e) \sum_{j=1}^{M} p_j \left(\frac{2^{-l_j}}{p_j} - 1 \right) \;=\; (\log e) \left(\sum_{j=1}^{M} 2^{-l_j} - \sum_{j=1}^{M} p_j \right) \le 0, \tag{2.9}$$

where the Kraft inequality and $\sum_j p_j = 1$ has been used. This establishes the left side of (2.7). The inequality in (2.9) is strict unless $2^{-l_j}/p_j = 1$, or equivalently $l_j = -\log p_j$, for all j. For integer l_j, this can be satisfied with equality if and only if p_j is an integer power of 2 for all j. For arbitrary real values of l_j, this proves that (2.5) minimizes (2.3) without the integer constraint, thus verifying (2.6.)

To complete the proof, it will be shown that a prefix-free code exists with $\overline{L} < \mathsf{H}[X]+1$. Choose the codeword lengths to be

$$l_j = \lceil -\log p_j \rceil ,$$

where the ceiling notation $\lceil u \rceil$ denotes the smallest integer less than or equal to u. With this choice,

$$-\log p_j \le l_j < -\log p_j + 1. \tag{2.10}$$

Since the left side of (2.10) is equivalent to $2^{-l_j} \le p_j$, the Kraft inequality is satisfied:

$$\sum_j 2^{-l_j} \le \sum_j p_j = 1.$$

Thus a prefix-free code exists with the above lengths. From the right side of (2.10), the expected codeword length of this code is upperbounded by

$$\overline{L} = \sum_j p_j l_j < \sum_j p_j \left(-\log p_j + 1\right) = \mathsf{H}[X] + 1.$$

Since $\overline{L}_{\min} \le \overline{L}$, $\overline{L}_{\min} < \mathsf{H}[X] + 1$, completing the proof. □

Both the proof above and the noninteger minimization in (2.6) suggest that the optimal length of a codeword for a source symbol of probability p_j should be approximately $-\log p_j$. This is not quite true, because, for example, if $M = 2$ and $p_1 = 2^{-20}$, $p_2 = 1-2^{-20}$, then $-\log p_1 = 20$, but the optimal l_1 is 1. However, the last part of the above proof shows that if each l_i is chosen as an integer approximation to $-\log p_i$, then $\overline{L}$ is at worst within one bit of $\mathsf{H}[X]$.

For sources with a small number of symbols, the upper bound in the theorem appears to be too loose to have any value. When these same arguments are applied later to long blocks of source symbols, however, the theorem leads directly to the source coding theorem.

2.5.3 Huffman's algorithm for optimal source codes

In the very early days of information theory, a number of heuristic algorithms were suggested for choosing codeword lengths l_j to approximate $-\log p_j$. Both Claude Shannon and Robert Fano had suggested such heuristic algorithms by 1948. It was conjectured at that time that, since this was an integer optimization problem, its optimal solution would be quite difficult. It was quite a surprise therefore when David Huffman [13] came up with a very simple and straightforward algorithm for constructing optimal (in the sense of minimal $\overline{L}$) prefix-free codes. Huffman developed the algorithm in 1950 as a term paper in Robert Fano's information theory class at MIT.

Huffman's trick, in today's jargon, was to "think outside the box." He ignored the Kraft inequality, and looked at the binary code tree to establish properties that an optimal prefix-free code should have. After discovering a few simple properties, he realized that they led to a simple recursive procedure for constructing an optimal code.

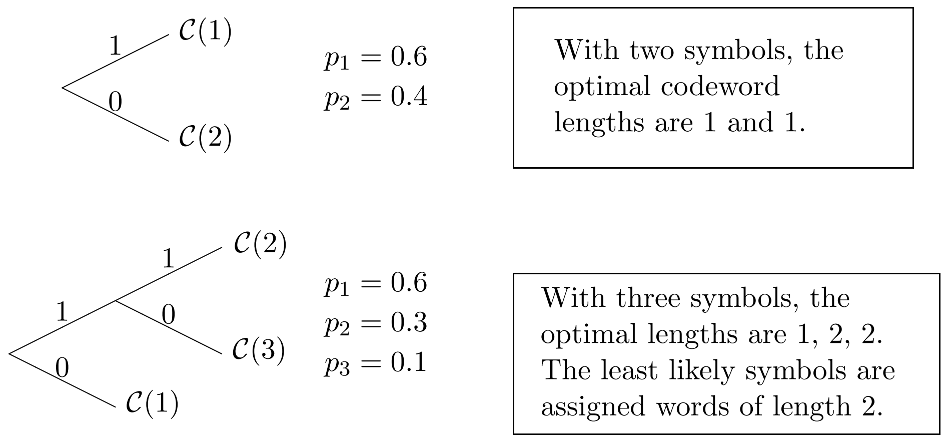

1
$\mathcal{C}(1)$
0
$\mathcal{C}(2)$
$p_1 = 0.6$
$p_2 = 0.4$
With two symbols, the optimal codeword lengths are 1 and 1.
1
$\mathcal{C}(2)$
1
0
$\mathcal{C}(3)$
0
$\mathcal{C}(1)$
$p_1 = 0.6$
$p_2 = 0.3$
$p_3 = 0.1$
With three symbols, the optimal lengths are 1, 2, 2. The least likely symbols are assigned words of length 2.

Figure 2.8: Some simple optimal codes.

The simple examples in Figure 2.8 illustrate some key properties of optimal codes. After stating these properties precisely, the Huffman algorithm will be almost obvious.

The property of the length assignments in the three-word example above can be generalized as follows: the longer the codeword, the less probable the corresponding symbol must be. More precisely:

Lemma 2.5.1. *Optimal codes have the property that if $p_i > p_j$, then $l_i \le l_j$.*

Proof: Assume to the contrary that a code has $p_i > p_j$ and $l_i > l_j$. The terms involving symbols i and j in $\overline{L}$ are $p_il_i + p_jl_j$. If the two code words are interchanged, thus interchanging l_i and l_j, this sum decreases, *i.e.*,

$$(p_il_i{+}p_jl_j) - (p_il_j{+}p_jl_i) = (p_i - p_j)(l_i - l_j) > 0.$$

Thus $\overline{L}$ decreases, so any code with $p_i > p_j$ and $l_i > l_j$ is nonoptimal. □

An even simpler property of an optimal code is as follows:

Lemma 2.5.2. *Optimal prefix-free codes have the property that the associated code tree is full.*

Proof: If the tree is not full, then a codeword length could be reduced (see Figures 2.2 and 2.3). □

Define the *sibling* of a codeword as the binary string that differs from the codeword in only the final digit. A sibling in a full code tree can be either a codeword or an intermediate node of the tree.

Lemma 2.5.3. *Optimal prefix-free codes have the property that, for each of the longest codewords in the code, the sibling of that codeword is another longest codeword.*

Proof: A sibling of a codeword of maximal length cannot be a prefix of a longer codeword. Since it cannot be an intermediate node of the tree, it must be a codeword. □

For notational convenience, assume that the $M = |\mathcal{X}|$ symbols in the alphabet are ordered so that $p_1 \ge p_2 \ge \cdots \ge p_M$.

Lemma 2.5.4. *Let X be a random symbol with a pmf satisfying $p_1 \geq p_2 \geq \cdots \geq p_M$. There is an optimal prefix-free code for X in which the codewords for $M-1$ and M are siblings and have maximal length within the code.*

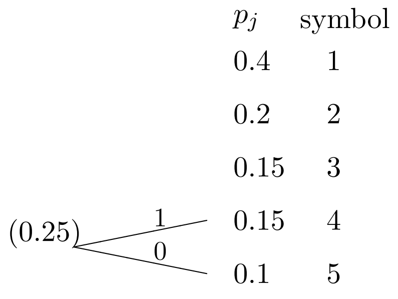
Proof: There are finitely many codes satisfying the Kraft inequality with equality,[11] so consider a particular one that is optimal. If $p_M < p_j$ for each $j < M$, then, from Lemma 2.5.1, $l_M \geq l_j$ for each and l_M has maximal length. If $p_M = p_j$ for one or more $j < M$, then l_j must be maximal for at least one such j. Then if l_M is not maximal, $\mathcal{C}(j)$ and $\mathcal{C}(M)$ can be interchanged with no loss of optimality, after which l_M is maximal. Now if $\mathcal{C}(k)$ is the sibling of $\mathcal{C}(M)$ in this optimal code, then l_k also has maximal length. By the argument above, $\mathcal{C}(M-1)$ can then be exchanged with $\mathcal{C}(k)$ with no loss of optimality. □

The Huffman algorithm chooses an optimal code tree by starting with the two least likely symbols, specifically M and $M-1$, and constraining them to be siblings in the yet unknown code tree. It makes no difference which sibling ends in 1 and which in 0. How is the rest of the tree to be chosen?

If the above pair of siblings is removed from the yet unknown tree, the rest of the tree must contain $M-1$ leaves, namely the $M-2$ leaves for the original first $M-2$ symbols, and the parent node of the removed siblings. The probability p'_{M-1} associated with this new leaf is taken as $p_{M-1} + p_M$. This tree of $M-1$ leaves is viewed as a code for a reduced random symbol X' with a reduced set of probabilities given as $p_1, \dots, p_{M-2}$ for the original first $M-2$ symbols and p'_{M-1} for the new symbol $M-1$.

To complete the algorithm, an optimal code is constructed for X'. It will be shown that an optimal code for X can be generated by constructing an optimal code for X', and then grafting siblings onto the leaf corresponding to symbol $M-1$. Assuming this fact for the moment, the problem of constructing an optimal M-ary code has been replaced with constructing an optimal $M-1$-ary code. This can be further reduced by applying the same procedure to the $M-1$-ary random symbol, and so forth down to a binary symbol for which the optimal code is obvious.

The following example in Figures 2.9 to 2.11 will make the entire procedure obvious. It starts with a random symbol X with probabilities $\{0.4, 0.2, 0.15, 0.15, 0.1\}$ and generates the reduced random symbol X' in Figure 2.9. The subsequent reductions are shown in Figures 2.10 and 2.11.



p_j	symbol
0.4	1
0.2	2
0.15	3
0.15	4
0.1	5

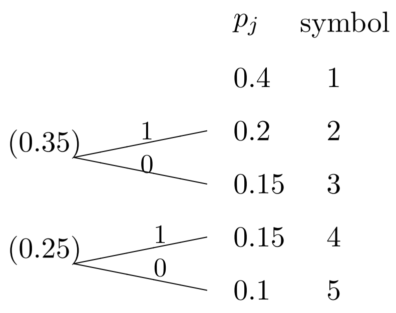
The two least likely symbols, 4 and 5 have been combined as siblings. The reduced set of probabilities then becomes {0.4, 0.2, 0.15, 0.25}.

Figure 2.9: Step 1 of the Huffman algorithm; finding X' from X

.

Another example using a different set of probabilities and leading to a different set of codeword lengths is given in Figure 2.12:

[11]Exercise 2.10 proves this for those who enjoy such things.



The two least likely symbols in the reduced set, with probabilities 0.15 and 0.2, have been combined as siblings. The reduced set of probabilities then becomes {0.4, 0.35, 0.25}.

Figure 2.10: Finding X'' from X'.

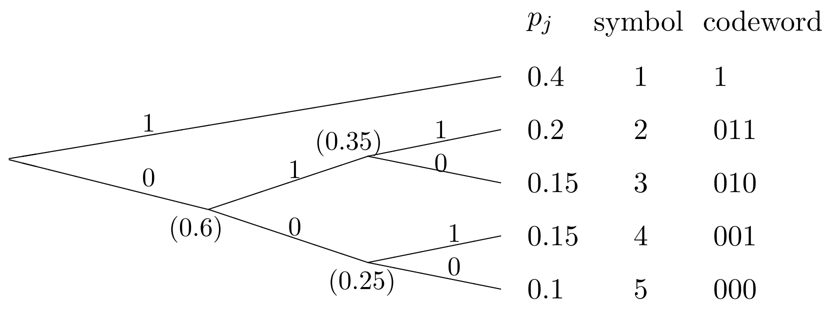


Figure 2.11: The completed Huffman code.

The only thing remaining to show that the Huffman algorithm constructs optimal codes is to show that an optimal code for the reduced random symbol X' yields an optimal code for X. Consider Figure 2.13, which shows the code tree for X' corresponding to X in Figure 2.12.

Note that Figures 2.12 and 2.13 differ in that $\mathcal{C}(4)$ and $\mathcal{C}(5)$, each of length 3 in Figure 2.12, have been replaced by a single codeword of length 2 in Figure 2.13. The probability of that single symbol is the sum of the two probabilities in Figure 2.12. Thus the expected codeword length for Figure 2.12 is that for Figure 2.13, increased by $p_4 + p_5$. This accounts for the fact that $\mathcal{C}(4)$ and $\mathcal{C}(5)$ have lengths one greater than their parent node.

In general, comparing the expected length $\overline{L}'$ of *any* code for X' and the corresponding $\overline{L}$ of the code generated by extending $\mathcal{C}'(M-1)$ in the code for X' into two siblings for $M-1$ and M, it is seen that

$$\overline{L} = \overline{L}' + p_{M-1} + p_M.$$

This relationship holds for all codes for X in which $\mathcal{C}(M-1)$ and $\mathcal{C}(M)$ are siblings (which includes at least one optimal code). This proves that $\overline{L}$ is minimized by minimizing $\overline{L}'$, and also shows that $\overline{L}_{\min} = \overline{L}'_{\min} + p_{M-1} + p_M$. This completes the proof of the optimality of the Huffman algorithm.

It is curious that neither the Huffman algorithm nor its proof of optimality give any indication of the entropy bounds, $\mathsf{H}[X] \le \overline{L}_{\min} < \mathsf{H}[X] + 1$. Similarly, the entropy bounds do not suggest the Huffman algorithm. One is useful in finding an optimal code; the other provides insightful performance bounds.

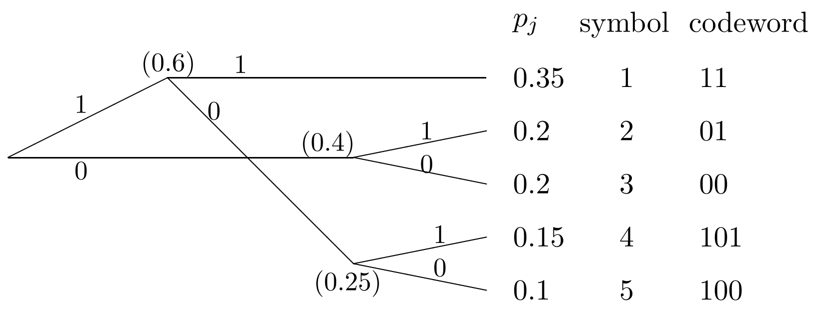


Figure 2.12: Completed Huffman code for a different set of probabilities.

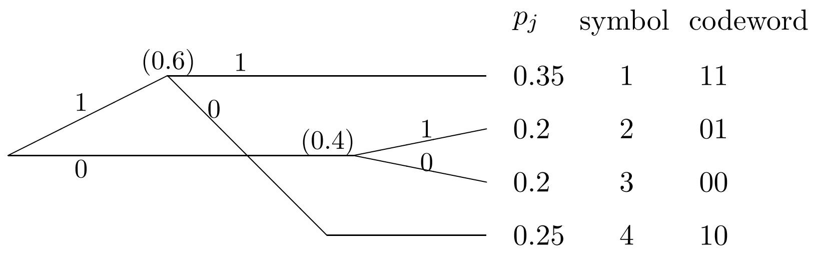


Figure 2.13: Completed reduced Huffman code for Figure 2.12.

As an example of the extent to which the optimal lengths approximate $-\log p_j$, the source probabilities in Figure 2.11 are $\{0.40, 0.20, 0.15, 0.15, 0.10\}$, so $-\log p_j$ takes the set of values $\{1.32, 2.32, 2.74, 2.74, 3.32\}$ bits; this approximates the lengths $\{1, 3, 3, 3, 3\}$ of the optimal code quite well. Similarly, the entropy is $\mathsf{H}[X] = 2.15$ bits/symbol and $\overline{L}_{\min} = 2.2$ bits/symbol, quite close to $\mathsf{H}[X]$. However, it would be difficult to guess these optimal lengths, even in such a simple case, without the algorithm.

For the example of Figure 2.12, the source probabilities are $\{0.35, 0.20, 0.20, 0.15, 0.10\}$, the values of $-\log p_i$ are $\{1.51, 2.32, 2.32, 2.74, 3.32\}$, and the entropy is $\mathsf{H}[X] = 2.20$. This is not very different from Figure 2.11. However, the Huffman code now has lengths $\{2, 2, 2, 3, 3\}$ and average length $\overline{L} = 2.25$ bits/symbol. (The code of Figure 2.11 has average length $\overline{L} = 2.30$ for these source probabilities.) It would be hard to predict these perturbations without carrying out the algorithm.

2.6 Entropy and fixed-to-variable-length codes

Entropy is now studied in more detail, both to better understand the entropy bounds and to understand the entropy of n-tuples of successive source letters.

The entropy $\mathsf{H}[X]$ is a fundamental measure of the randomness of a random symbol X. It has many important properties. The property of greatest interest here is that it is the smallest expected number $\overline{L}$ of bits per source symbol required to map the sequence of source symbols into a bit sequence in a uniquely decodable way. This will soon be demonstrated by generalizing the variable-length codes of the last few sections to codes in which multiple source symbols are encoded together. First, however, several other properties of entropy are derived.

Definition: The *entropy* of a discrete random symbol[12] X with alphabet $\mathcal{X}$ is

$$\mathsf{H}[X] = \sum_{x\in\mathcal{X}} p_X(x)\log\frac{1}{p_X(x)} = -\sum_{x\in\mathcal{X}} p_X(x)\log p_X(x). \tag{2.11}$$

Using logarithms to the base 2, the units of $\mathsf{H}[X]$ are *bits/symbol.* If the base of the logarithm is e, then the units of $\mathsf{H}[X]$ are called nats/symbol. Conversion is easy; just remember that $\log y = (\ln y)/(\ln 2)$ or $\ln y = (\log y)/(\log e)$, both of which follow from $y = e^{\ln y} = 2^{\log y}$ by taking logarithms. Thus using another base for the logarithm just changes the numerical units of entropy by a scale factor.

Note that the entropy $\mathsf{H}[X]$ of a discrete random symbol X depends on the probabilities of the different outcomes of X, but not on the names of the outcomes. Thus, for example, the entropy of a random symbol taking the values GREEN, BLUE, and RED with probabilities $0.2, 0.3, 0.5$, respectively, is the same as the entropy of a random symbol taking on the values SUNDAY, MONDAY, FRIDAY with the same probabilities $0.2, 0.3, 0.5$.

The entropy $\mathsf{H}[X]$ is also called the *uncertainty* of X, meaning that it is a measure of the randomness of X. Note that entropy is the expected value of the rv $\log(1/p_X(X))$. This random variable is called the *log pmf* rv.[13] Thus the entropy is the expected value of the log pmf rv.

Some properties of entropy:

- For any discrete random symbol X, $\mathsf{H}[X] \geq 0$. This follows because $p_X(x) \leq 1$, so $\log(1/p_X(x)) \geq 0$. The result follows from (2.11).
- $\mathsf{H}[X] = 0$ if and only if X is deterministic. This follows since $p_X(x)\log(1/p_X(x)) = 0$ if and only if $p_X(x)$ equals 0 or 1.
- The entropy of an equiprobable random symbol X with an alphabet $\mathcal{X}$ of size M is $\mathsf{H}[X] = \log M$. This follows because, if $p_X(x) = 1/M$ for all $x \in \mathcal{X}$, then

  $$\mathsf{H}[X] = \sum_{x\in\mathcal{X}} \frac{1}{M}\log M = \log M.$$

  In this case, the rv $-\log p_X(X)$ has the constant value $\log M$.
- More generally, the entropy $\mathsf{H}[X]$ of a random symbol X defined on an alphabet $\mathcal{X}$ of size M satisfies $\mathsf{H}[X] \leq \log M$, with equality only in the equiprobable case. To see this, note that

  $$\begin{aligned}\mathsf{H}[X] - \log M &= \sum_{x\in\mathcal{X}} p_X(x)\left[\log\frac{1}{p_X(x)} - \log M\right] = \sum_{x\in\mathcal{X}} p_X(x)\left[\log\frac{1}{Mp_X(x)}\right] \\ &\leq (\log e)\sum_{x\in\mathcal{X}} p_X(x)\left[\frac{1}{Mp_X(x)} - 1\right] = 0,\end{aligned}$$

[12] If one wishes to consider discrete random symbols with one or more symbols of zero probability, one can still use this formula by recognizing that $\lim_{p\to 0} p\log(1/p) = 0$ and then defining $0\log 1/0$ as 0 in (2.11). Exercise 2.18 illustrates the effect of zero probability symbols in a variable-length prefix code.

[13] This rv is often called *self information* or *surprise*, or *uncertainty.* It bears some resemblance to the ordinary meaning of these terms, but historically this has caused much more confusion than enlightenment. Log pmf, on the other hand, emphasizes what is useful here

This uses the inequality $\log u \leq (\log e)(u-1)$ (after omitting any terms for which $p_X(x) = 0$). For equality, it is necessary that $p_X(x) = 1/M$ for all $x \in \mathcal{X}$.

In summary, of all random symbols X defined on a given finite alphabet $\mathcal{X}$, the highest entropy occurs in the equiprobable case, namely $\mathsf{H}[X] = \log M$, and the lowest occurs in the deterministic case, namely $\mathsf{H}[X] = 0$. This supports the intuition that the entropy of a random symbol X is a measure of its randomness.

For any pair of discrete random symbols X and Y, XY is another random symbol. The sample values of XY are the set of all pairs xy, $x \in \mathcal{X}, y \in \mathcal{Y}$ and the probability of each sample value xy is $p_{XY}(x,y)$. An important property of entropy is that if X and Y are independent discrete random symbols, then $\mathsf{H}[XY] = \mathsf{H}[X] + \mathsf{H}[Y]$. This follows from:

$$\begin{aligned} \mathsf{H}[XY] &= -\sum_{\mathcal{X}\times\mathcal{Y}} p_{XY}(x,y)\log p_{XY}(x,y) \\ &= -\sum_{\mathcal{X}\times\mathcal{Y}} p_X(x)p_Y(y)\left(\log p_X(x) + \log p_Y(y)\right) = \mathsf{H}[X] + \mathsf{H}[Y]. \end{aligned} \qquad (2.12)$$

Extending this to n random symbols, the entropy of a random symbol $\boldsymbol{X}^n$ corresponding to a block of n iid outputs from a discrete memoryless source is $\mathsf{H}[\boldsymbol{X}^n] = n\mathsf{H}[X]$; *i.e.*, each symbol increments the entropy of the block by $\mathsf{H}[X]$ bits.

2.6.1 Fixed-to-variable-length codes

Recall that in Section 2.2 the sequence of symbols from the source was segmented into successive blocks of n symbols which were then encoded. Each such block was a discrete random symbol in its own right, and thus could be encoded as in the single-symbol case. It was seen that by making n large, fixed-length codes could be constructed in which the number $\overline{L}$ of encoded bits per source symbol approached $\log M$ as closely as desired.

The same approach is now taken for variable-length coding of discrete memoryless sources. A block of n source symbols, $X_1, X_2, \ldots, X_n$ has entropy $\mathsf{H}[\boldsymbol{X}^n] = n\mathsf{H}[X]$. Such a block is a random symbol in its own right and can be encoded using a variable-length prefix-free code. This provides a fixed-to-variable-length code, mapping n-tuples of source symbols to variable-length binary sequences. It will be shown that the expected number $\overline{L}$ of encoded bits per source symbol can be made as close to $\mathsf{H}[X]$ as desired.

Surprisingly, this result is very simple. Let $\mathsf{E}[L(\boldsymbol{X}^n)]$ be the expected length of a variable-length prefix-free code for $\boldsymbol{X}^n$. Denote the minimum expected length of any prefix-free code for $\boldsymbol{X}^n$ by $\mathsf{E}[L(\boldsymbol{X}^n)]_{\min}$. Theorem 2.5.1 then applies. Using (2.7),

$$\mathsf{H}[\boldsymbol{X}^n] \leq \mathsf{E}[L(\boldsymbol{X}^n)]_{\min} < \mathsf{H}[\boldsymbol{X}^n] + 1. \qquad (2.13)$$

Define $\overline{L}_{\min,n} = \frac{\mathsf{E}[L(\boldsymbol{X}^n)]_{\min}}{n}$; *i.e.*, $\overline{L}_{\min,n}$ is the minimum number of bits per source symbol over all prefix-free codes for $\boldsymbol{X}^n$. From (2.13),

$$\mathsf{H}[X] \leq \overline{L}_{\min,n} < \mathsf{H}[X] + \frac{1}{n}. \qquad (2.14)$$

This simple result establishes the following important theorem:

Theorem 2.6.1 (Prefix-free source coding theorem). *For any discrete memoryless source with entropy $\mathsf{H}[X]$, and any integer $n \geq 1$, there exists a prefix-free encoding of source n-tuples for which the expected codeword length per source symbol $\overline{L}$ is at most $\mathsf{H}[X] + 1/n$. Furthermore, no prefix-free encoding of fixed-length source blocks of any length n results in an expected codeword length $\overline{L}$ less than $\mathsf{H}[X]$.*

This theorem gives considerable significance to the entropy $\mathsf{H}[X]$ of a discrete memoryless source: $\mathsf{H}[X]$ is the minimum expected number $\overline{L}$ of bits per source symbol that can be achieved by fixed-to-variable-length prefix-free codes.

There are two potential questions about the significance of the theorem. First, is it possible to find uniquely-decodable codes other than prefix-free codes for which $\overline{L}$ is less than $\mathsf{H}[X]$? Second, is it possible to further reduce $\overline{L}$ by using variable-to-variable-length codes?

For example, if a binary source has $p_1 = 10^{-6}$ and $p_0 = 1 - 10^{-6}$, fixed-to-variable-length codes must use remarkably long n-tuples of source symbols to approach the entropy bound. Run-length coding, which is an example of variable-to-variable-length coding, is a more sensible approach in this case: the source is first encoded into a sequence representing the number of source 0's between each 1, and then this sequence of integers is encoded. This coding technique is further developed in Exercise 2.23.

The next section strengthens Theorem 2.6.1, showing that $\mathsf{H}[X]$ is indeed a lower bound to $\overline{L}$ over all uniquely-decodable encoding techniques.

2.7 The AEP and the source coding theorems

We first review the weak[14] law of large numbers (WLLN) for sequences of iid rv's. Applying the WLLN to a particular iid sequence, we will establish a form of the remarkable asymptotic equipartition property (AEP).

Crudely, the AEP says that, given a very long string of n iid discrete random symbols $X_1, \dots, X_n$, there exists a "typical set" of sample strings $(x_1, \dots, x_n)$ whose aggregate probability is almost 1. There are roughly $2^{n\mathsf{H}[X]}$ typical strings of length n, and each has a probability roughly equal to $2^{-n\mathsf{H}[X]}$. We will have to be careful about what the words "almost" and "roughly" mean here.

The AEP will give us a fundamental understanding not only of source coding for discrete memoryless sources, but also of the probabilistic structure of such sources and the meaning of entropy. The AEP will show us why general types of source encoders, such as variable-to-variable-length encoders, cannot have a strictly smaller expected length per source symbol than the best fixed-to-variable-length prefix-free codes for discrete memoryless sources.

[14]The word *weak* is something of a misnomer, since this is one of the most useful results in probability theory. There is also a strong law of large numbers; the difference lies in the limiting behavior of an infinite sequence of rv's, but this difference is not relevant here. The weak law applies in some cases where the strong law does not, but this also is not relevant here.

2.7.1 The weak law of large numbers

Let $Y_1, Y_2, \dots,$ be a sequence of iid rv's. Let $\overline{Y}$ and σ_Y^2 be the mean and variance of each Y_j. Define the *sample average* A_Y^n of $Y_1, \dots, Y_n$ as

$$A_Y^n = \frac{S_Y^n}{n} \quad where \quad S_Y^n = Y_1 + \cdots + Y_n.$$

The sample average A_Y^n is itself an rv, whereas, of course, the mean $\overline{Y}$ is simply a real number. Since the sum S_Y^n has mean $n\overline{Y}$ and variance $n\sigma_Y^2$, the sample average A_Y^n has mean $\mathsf{E}[A_Y^n] = \overline{Y}$ and variance $\sigma^2_{A_Y^n} = \sigma^2_{S_Y^n}/n^2 = \sigma_Y^2/n$. It is important to understand that the variance of the sum *increases* with n and the variance of the normalized sum (the sample average, A_Y^n), *decreases* with n.

The Chebyshev inequality states that if $\sigma_X^2 < \infty$ for an rv X, then, $\Pr\{|X - \overline{X}| \geq \varepsilon\} \leq \sigma_X^2/\varepsilon^2$ for any $\varepsilon > 0$ (see Exercise 2.3 or any text on probability such as [2] or [24]). Applying this inequality to A_Y^n yields the simplest form of the WLLN: for any $\varepsilon > 0$,

$$\Pr\{|A_Y^n - \overline{Y}| \geq \varepsilon\} \leq \frac{\sigma_Y^2}{n\varepsilon^2}. \tag{2.15}$$

This is illustrated in Figure 2.14.

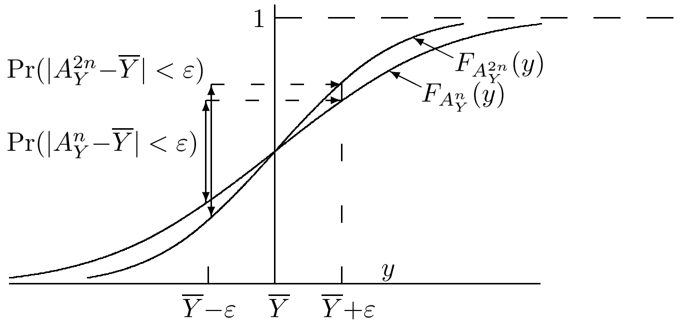


Figure 2.14: Sketch of the distribution function of the sample average for different n. As n increases, the distribution function approaches a unit step at $\overline{Y}$. The closeness to a step within $\overline{Y} \pm \varepsilon$ is upperbounded by (2.15).

Since the right side of (2.15) approaches 0 with increasing n for any fixed $\varepsilon > 0$,

$$\lim_{n\to\infty} \Pr\{|A_Y^n - \overline{Y}| \geq \varepsilon\} = 0. \tag{2.16}$$

For large n, (2.16) says that $A_Y^n - \overline{Y}$ is small with high probability. It does not say that $A_Y^n = \overline{Y}$ with high probability (or even nonzero probability), and it does not say that $\Pr(|A_Y^n - \overline{Y}| \geq \varepsilon) = 0$. As illustrated in Figure 2.14, both a nonzero ε and a nonzero probability are required here, even though they can be made simultaneously as small as desired by increasing n.

In summary, the sample average A_Y^n is an rv whose mean $\overline{Y}$ is independent of n, but whose standard deviation $\sigma_Y/\sqrt{n}$ approaches 0 as $n \to \infty$. Therefore the distribution of the sample average becomes concentrated near $\overline{Y}$ as n increases. The WLLN is simply this concentration property, stated more precisely by either (2.15) or (2.16).

The WLLN, in the form of (2.16), applies much more generally than the simple case of iid rv's. In fact, (2.16) provides the central link between probability models and the real phenomena being modeled. One can observe the outcomes both for the model and reality, but probabilities are assigned only for the model. The WLLN, applied to a sequence of rv's in the model, and the concentration property (if it exists), applied to the corresponding real phenomenon, provide the basic check on whether the model corresponds reasonably to reality.

2.7.2 The asymptotic equipartition property

This section starts with a sequence of iid random symbols and defines a sequence of random variables (rv's) as functions of those symbols. The WLLN, applied to these rv's, will permit the classification of sample sequences of symbols as being 'typical' or not, and then lead to the results alluded to earlier.

Let $X_1, X_2, \dots$ be a sequence of iid discrete random symbols with a common pmf $p_X(x)>0$, $x\in\mathcal{X}$. For each symbol x in the alphabet $\mathcal{X}$, let $w(x) = -\log p_X(x)$. For each X_k in the sequence, define $W(X_k)$ to be the rv that takes the value $w(x)$ for $X_k = x$. Then $W(X_1), W(X_2), \dots$ is a sequence of iid discrete rv's, each with mean

$$\mathsf{E}[W(X_k)] = -\sum_{x\in\mathcal{X}} p_X(x)\log p_X(x) = \mathsf{H}[X], \tag{2.17}$$

where $\mathsf{H}[X]$ is the entropy of the random symbol X.

The rv $W(X_k)$ is the *log pmf* of X_k and the entropy of X_k is the mean of $W(X_k)$.

The most important property of the log pmf for iid random symbols comes from observing, for example, that for the event $X_1 = x_1, X_2 = x_2$, the outcome for $W(X_1) + W(X_2)$ is

$$w(x_1) + w(x_2) = -\log p_X(x_1) - \log p_X(x_2) = -\log\{p_{X_1X_2}(x_1x_2)\}. \tag{2.18}$$

In other words, the joint pmf for independent random symbols is the product of the individual pmf's, and therefore *the log of the joint pmf is the sum of the logs of the individual pmf's.*

We can generalize (2.18) to a string of n random symbols, $\boldsymbol{X}^n = (X_1, \dots, X_n)$. For an event $\boldsymbol{X}^n = \boldsymbol{x}^n$ where $\boldsymbol{x}^n = (x_1, \dots, x_n)$, the outcome for the sum $W(X_1) + \cdots + W(X_n)$ is

$$\sum\nolimits_{k=1}^{n} w(x_k) = -\sum\nolimits_{k=1}^{n} \log p_X(x_k) = -\log p_{\boldsymbol{X}^n}(\boldsymbol{x}^n). \tag{2.19}$$

The WLLN can now be applied to the sample average of the log pmfs. Let

$$A_W^n = \frac{W(X_1) + \cdots + W(X_n)}{n} = \frac{-\log p_{\boldsymbol{X}^n}(\boldsymbol{X}^n)}{n} \tag{2.20}$$

be the sample average of the log pmf.

From (2.15), it follows that

$$\Pr\Big(\big|A_W^n - \mathsf{E}[W(X)]\big| \geq \varepsilon\Big) \leq \frac{\sigma_W^2}{n\varepsilon^2}. \tag{2.21}$$

Substituting (2.17) and (2.20) into (2.21),

$$\Pr\left(\left|\frac{-\log p_{\boldsymbol{X}^n}(\boldsymbol{X}^n)}{n} - \mathsf{H}[X]\right| \geq \varepsilon\right) \leq \frac{\sigma_W^2}{n\varepsilon^2}. \tag{2.22}$$

In order to interpret this result, define the *typical set* T_ε^n for any $\varepsilon > 0$ as

$$T_\varepsilon^n = \left\{ \boldsymbol{x}^n : \left| \frac{-\log p_{\boldsymbol{X}^n}(\boldsymbol{x}^n)}{n} - \mathsf{H}[X] \right| < \varepsilon \right\}. \tag{2.23}$$

Thus T_ε^n is the set of source strings of length n for which the sample average of the log pmf is within ε of its mean $\mathsf{H}[X]$. Eq. (2.22) then states that the aggregrate probability of all strings of length n not in T_ε^n is at most $\sigma_W^2/(n\varepsilon^2)$. Thus,

$$\Pr(\boldsymbol{X}^n \in T_\varepsilon^n) \geq 1 - \frac{\sigma_W^2}{n\varepsilon^2}. \tag{2.24}$$

As n increases, the aggregate probability of T_ε^n approaches 1 for any given $\varepsilon > 0$, so T_ε^n is certainly a typical set of source strings. This is illustrated in Figure 2.15.

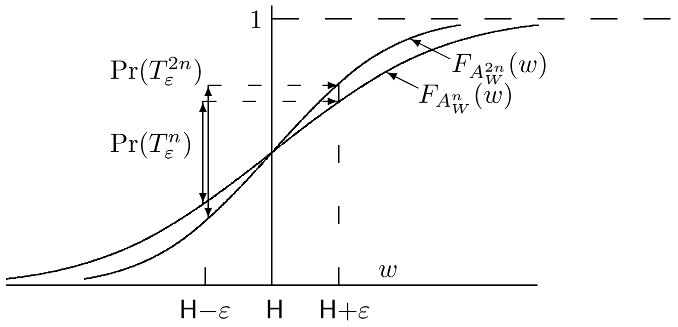


Figure 2.15: Sketch of the distribution function of the sample average log pmf. As n increases, the distribution function approaches a unit step at H. The typical set is the set of sample strings of length n for which the sample average log pmf stays within ε of H; as illustrated, its probability approaches 1 as $n \to \infty$.

Rewrite (2.23) in the form

$$T_\varepsilon^n = \left\{ \boldsymbol{x}^n : \; n(\mathsf{H}[X] - \varepsilon) < -\log p_{\boldsymbol{X}^n}(\boldsymbol{x}^n) < n(\mathsf{H}[X] + \varepsilon) \right\}.$$

Multiplying by -1 and exponentiating,

$$T_\varepsilon^n = \left\{ \boldsymbol{x}^n : \; 2^{-n(\mathsf{H}[X]+\varepsilon)} < p_{\boldsymbol{X}^n}(\boldsymbol{x}^n) < 2^{-n(\mathsf{H}[X]-\varepsilon)} \right\}. \tag{2.25}$$

Eq. (2.25) has the intuitive connotation that the n-strings in T_ε^n are approximately equiprobable. This is the same kind of approximation that one would use in saying that $10^{-1001} \approx 10^{-1000}$; these numbers differ by a factor of 10, but for such small numbers it makes sense to compare the exponents rather than the numbers themselves. In the same way, the ratio of the upper to lower bound in (2.25) is $2^{2\varepsilon n}$, which grows unboundedly with n for fixed ε. However, as seen in (2.23), $-\frac{1}{n}\log p_{\boldsymbol{X}^n}(\boldsymbol{x}^n)$ is approximately equal to $\mathsf{H}[X]$ for all $\boldsymbol{x}^n \in T_\varepsilon^n$. This is the important notion, and it does no harm to think of the n-strings in T_ε^n as being approximately equiprobable.

The set of all n-strings of source symbols is thus separated into the typical set T_ε^n and the complementary atypical set $(T_\varepsilon^n)^c$. The atypical set has aggregate probability no greater than $\sigma_W^2/(n\varepsilon^2)$, and the elements of the typical set are approximately equiprobable (in this peculiar sense), each with probability about $2^{-n\mathsf{H}[X]}$.

The typical set T_ε^n depends on the choice of ε. As ε decreases, the equiprobable approximation (2.25) becomes tighter, but the bound (2.24) on the probability of the typical set is further from 1. As n increases, however, ε can be slowly decreased, thus bringing the probability of the typical set closer to 1 and simultaneously tightening the bounds on equiprobable strings.

Let us now estimate the number of elements $|T_\varepsilon^n|$ in the typical set. Since $p_{\boldsymbol{X}^n}(\boldsymbol{x}^n) > 2^{-n(\mathsf{H}[X]+\varepsilon)}$ for each $\boldsymbol{x}^n \in T_\varepsilon^n$,

$$1 \geq \sum_{\boldsymbol{x}^n \in T_\varepsilon^n} p_{\boldsymbol{X}^n}(\boldsymbol{x}^n) > |T_\varepsilon^n|\, 2^{-n(\mathsf{H}[X]+\varepsilon)}.$$

This implies that $|T_\varepsilon^n| < 2^{n(\mathsf{H}[X]+\varepsilon)}$. In other words, since each $\boldsymbol{x}^n \in T_\varepsilon^n$ contributes at least $2^{-n(\mathsf{H}[X]+\varepsilon)}$ to the probability of T_ε^n, the number of these contributions can be no greater than $2^{n(\mathsf{H}[X]+\varepsilon)}$.

Conversely, since $\Pr(T_\varepsilon^n) \geq 1 - \sigma_W^2/(n\varepsilon^2)$, $|T_\varepsilon^n|$ can be lower bounded by

$$1 - \frac{\sigma_W^2}{n\varepsilon^2} \leq \sum_{\boldsymbol{x}^n \in T_\varepsilon^n} p_{\boldsymbol{X}^n}(\boldsymbol{x}^n) < |T_\varepsilon^n| 2^{-n(\mathsf{H}[X]-\varepsilon)},$$

which implies $|T_\varepsilon^n| > [1 - \sigma_W^2/(n\varepsilon^2)]2^{n(\mathsf{H}[X]-\varepsilon)}$. In summary,

$$\left(1 - \frac{\sigma_W^2}{n\varepsilon^2}\right) 2^{n(\mathsf{H}[X]-\varepsilon)} < |T_\varepsilon^n| < 2^{n(\mathsf{H}[X]+\varepsilon)}. \tag{2.26}$$

For large n, then, the typical set T_ε^n has aggregate probability approximately 1 and contains approximately $2^{n\mathsf{H}[X]}$ elements, each of which has probability approximately $2^{-n\mathsf{H}[X]}$. That is, asymptotically for very large n, the random symbol $\boldsymbol{X}^n$ resembles an equiprobable source with alphabet size $2^{n\mathsf{H}[X]}$.

The quantity $\sigma_W^2/(n\varepsilon^2)$ in many of the equations above is simply a particular upper bound to the probability of the atypical set. It becomes arbitrarily small as n increases for any fixed $\varepsilon > 0$. Thus it is insightful to simply replace this quantity with a real number δ; for any such $\delta > 0$ and any $\varepsilon > 0$, $\sigma_W^2/(n\varepsilon^2) \leq \delta$ for large enough n.

This set of results, summarized in the following theorem, is known as the *asymptotic equipartition property* (AEP).

Theorem 2.7.1 (Asymptotic equipartition property). *Let $\boldsymbol{X}^n$ be a string of n iid discrete random symbols $\{X_k; 1 \leq k \leq n\}$ each with entropy $H[X]$. For all $\delta > 0$ and all sufficiently large n, $\Pr(T_\varepsilon^n) \geq 1 - \delta$ and $|T_\varepsilon^n|$ is bounded by*

$$(1-\delta)2^{n(H[X]-\varepsilon)} < |T_\varepsilon^n| < 2^{n(H[X]+\varepsilon)}. \tag{2.27}$$

Finally, note that the total number of different strings of length n from a source with alphabet size M is M^n. For non-equiprobable sources, namely sources with $\mathsf{H}[X] < \log M$, the ratio of the number of typical strings to total strings is approximately $2^{-n(\log M - \mathsf{H}[X])}$, which approaches 0 exponentially with n. Thus, for large n, the great majority of n-strings are atypical. It may be somewhat surprising that this great majority counts for so little in probabilistic terms. As shown in Exercise 2.26, the most probable of the individual sequences are also atypical. There are too few of them, however, to have any significance.

We next consider source coding in the light of the AEP.

2.7.3 Source coding theorems

Motivated by the AEP, we can take the approach that an encoder operating on strings of n source symbols need only provide a codeword for each string $\boldsymbol{x}^n$ in the typical set T_ε^n. If a sequence $\boldsymbol{x}^n$ occurs that is not in T_ε^n, then a source coding failure is declared. Since the probability of $\boldsymbol{x}^n \notin T_\varepsilon^n$ can be made arbitrarily small by choosing n large enough, this situation is tolerable.

In this approach, since there are less than $2^{n(\mathsf{H}[X]+\varepsilon)}$ strings of length n in T_ε^n, the number of source codewords that need to be provided is fewer than $2^{n(\mathsf{H}[X]+\varepsilon)}$. Choosing fixed-length codewords of length $\lceil n(\mathsf{H}[X]+\varepsilon)\rceil$ is more than sufficient and even allows for an extra codeword, if desired, to indicate that a coding failure has occurred. In bits per source symbol, taking the ceiling function into account, $\overline{L} \leq \mathsf{H}[X]+\varepsilon+1/n$. Note that $\varepsilon > 0$ is arbitrary, and for any such ε, $\Pr\{\text{failure}\} \to 0$ as $n \to \infty$. This proves the following theorem:

Theorem 2.7.2 (Fixed-to-fixed-length source coding theorem). *For any discrete memoryless source with entropy $\mathsf{H}[X]$, any $\varepsilon > 0$, any $\delta > 0$, and any sufficiently large n, there is a fixed-to-fixed-length source code with $\Pr\{\text{failure}\} \leq \delta$ that maps blocks of n source symbols into fixed-length codewords of length $\overline{L} \leq \mathsf{H}[X] + \varepsilon + 1/n$ bits per source symbol.*

We saw in section 2.2 that the use of fixed-to-fixed-length source coding requires $\log M$ bits per source symbol if unique decodability is required (*i.e.*, no failures are allowed), and now we see that this is reduced to arbitrarily little more than $\mathsf{H}[X]$ bits per source symbol if arbitrarily rare failures are allowed. This is a good example of a situation where 'arbitrarily small $\delta > 0$' and 0 behave very differently.

There is also a converse to this theorem following from the other side of the AEP theorem. This says that the error probability approaches 1 for large n if strictly fewer than $\mathsf{H}[X]$ bits per source symbol are provided.

Theorem 2.7.3 (Converse for fixed-to-fixed-length codes). *Let $\boldsymbol{X}^n$ be a string of n iid discrete random symbols $\{X_k; 1 \leq k \leq n\}$, with entropy $\mathsf{H}[X]$ each. For any $\nu > 0$, let $\boldsymbol{X}^n$ be encoded into fixed-length codewords of length $\lfloor n(\mathsf{H}[X] - \nu)\rfloor$ bits. For every $\delta > 0$ and for all sufficiently large n given δ,*

$$\Pr\{\text{failure}\} > 1 - \delta - 2^{-\nu n/2}. \tag{2.28}$$

Proof: Apply the AEP, Theorem 2.7.1, with $\varepsilon = \nu/2$. Codewords can be provided for at most $2^{n(\mathsf{H}[X]-\nu)}$ typical source n-sequences, and from (2.25) each of these has a probability at most $2^{-n(\mathsf{H}[X]-\nu/2)}$. Thus the aggregate probability of typical sequences for which codewords are provided is at most $2^{-n\nu/2}$. From the AEP theorem, $\Pr\{T_\varepsilon^n\} \geq 1 - \delta$ is satisfied for large enough n. Codewords[15] can be provided for at most a subset of T_ε^n of probability $2^{-n\nu/2}$, and the remaining elements of T_ε^n must all lead to errors, thus yielding (2.28). □

In going from fixed-length codes of slightly more than $\mathsf{H}[X]$ bits per source symbol to codes of slightly less than $\mathsf{H}[X]$ bits per source symbol, the probability of failure goes from almost 0 to almost 1, and as n increases, those limits are approached more and more closely.

[15]Note that the proof allows codewords to be provided for atypical sequences; it simply says that a large portion of the typical set cannot be encoded.

2.7.4 The entropy bound for general classes of codes

We have seen that the expected number of encoded bits per source symbol is lower bounded by $\mathsf{H}[X]$ for iid sources using either fixed-to-fixed-length or fixed-to-variable-length codes. The details differ in the sense that very improbable sequences are simply dropped in fixed-length schemes but have abnormally long encodings, leading to buffer overflows, in variable-length schemes.

We now show that other types of codes, such as variable-to-fixed, variable-to-variable, and even more general codes are also subject to the entropy limit. Rather than describing the highly varied possible nature of these source codes, this will be shown by simply defining certain properties that the associated decoders must have. By doing this, it is also shown that yet undiscovered coding schemes must also be subject to the same limits. The fixed-to-fixed-length converse in the last subsection is the key to this.

For any encoder, there must be a decoder that maps the encoded bit sequence back into the source symbol sequence. For prefix-free codes on k-tuples of source symbols, the decoder waits for each variable length codeword to arrive, maps it into the corresponding k-tuple of source symbols, and then starts decoding for the next k-tuple. For fixed-to-fixed-length schemes, the decoder waits for a block of code symbols and then decodes the corresponding block of source symbols.

In general, the source produces a non-ending sequence $X_1, X_2, \dots$ of source letters which are encoded into a non-ending sequence of encoded binary digits. The decoder observes this encoded sequence and decodes source symbol X_n when enough bits have arrived to make a decision on it.

For any given coding and decoding scheme for a given iid source, define the rv D_n as the number of received bits that permit a decision on $\boldsymbol{X}^n = X_1, \dots, X_n$. This includes the possibility of coders and decoders for which some sample source strings are decodedeincorrectly or postponed infinitely. For these $\boldsymbol{x}^n$, the sample value of D_n is taken to be infinite. It is assumed, however, that all decisions are final in the sense that the decoder cannot decide on a particular $\boldsymbol{x}^n$ after observing an initial string of the encoded sequence and then change that decision after observing more of the encoded sequence. What we would like is a scheme in which decoding is correct with high probability and the sample value of the rate, D_n/n, is small with high probability. What the following theorem shows is that for large n, the sample rate can be strictly below the entropy only with vanishingly small probability. This then shows that the entropy lower bounds the data rate in this strong sense.

Theorem 2.7.4 (Converse for general coders/decoders for iid sources). *Let $\boldsymbol{X}^\infty$ be a sequence of discrete random symbols $\{X_k; 1 \le k \le \infty\}$. For each integer $n \ge 1$, let $\boldsymbol{X}^n$ be the first n of those symbols. For any given encoder and decoder, let D_n be the number of received bits at which the decoder can correctly decode $\boldsymbol{X}^n$. Then for any $\nu > 0$ and $\delta > 0$, and for any sufficiently large n given ν and δ,*

$$\Pr\{D_n \le n[H[X] - \nu]\} < \delta + 2^{-\nu n/2}. \tag{2.29}$$

Proof: For any sample value $\boldsymbol{x}^\infty$ of the source sequence, let $\boldsymbol{y}^\infty$ denote the encoded sequence. For any given integer $n \ge 1$, let $m = \lfloor n[\mathsf{H}[X] - \nu] \rfloor$. Suppose that $\boldsymbol{x}^n$ is decoded upon observation of $\boldsymbol{y}^j$ for some $j \le m$. Since decisions are final, there is only one source n-string, namely $\boldsymbol{x}^n$, that can be decoded by time $\boldsymbol{y}^m$ is observed. This means that out of the 2^m possible initial

m-strings from the encoder, there can be at most[16] 2^m n-strings from the source that be decoded from the observation of the first m encoded outputs. The aggregate probability of any set of 2^m source n-strings is bounded in Theorem 2.7.3, and (2.29) simply repeats that bound. □

2.8 Markov sources

The basic coding results for discrete memoryless sources have now been derived. Many of the results, in particular the Kraft inequality, the entropy bounds on expected length for uniquely-decodable codes, and the Huffman algorithm, do not depend on the independence of successive source symbols.

In this section, these results are extended to sources defined in terms of finite-state Markov chains. The state of the Markov chain[17] is used to represent the "memory" of the source. Labels on the transitions between states are used to represent the next symbol out of the source. Thus, for example, the state could be the previous symbol from the source, or could be the previous 300 symbols. It is possible to model as much memory as desired while staying in the regime of finite-state Markov chains.

Example 2.8.1. Consider a binary source with outputs $X_1, X_2, \ldots$. Assume that the symbol probabilities for X_m are conditioned on X_{k-2} and X_{k-1} but are independent of all previous symbols given these past 2 symbols. This pair of previous symbols is modeled by a *state* S_{k-1}. The alphabet of possible states is then the set of binary pairs, $\mathcal{S} = \{[00], [01], [10], [11]\}$. In Figure 2.16, the states are represented as the nodes of the graph representing the Markov chain, and the source outputs are labels on the graph transitions. Note, for example, that from the state $S_{k-1} = [01]$ (representing $X_{k-2}{=}0, X_{k-1}{=}1$), the output $X_k{=}1$ causes a transition to $S_k = [11]$ (representing $X_{k-1}{=}1, X_k{=}1$). The chain is assumed to start at time 0 in a state S_0 given by some arbitrary pmf.

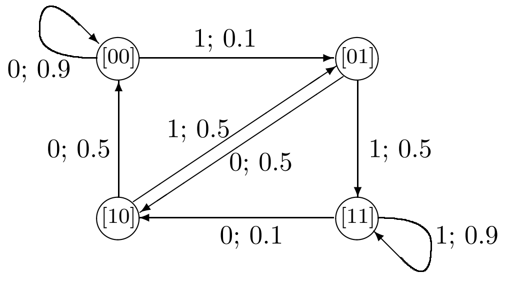


Figure 2.16: Markov source: Each transition $s' \to s$ is labeled by the corresponding source output and the transition probability $\Pr\{S_k = s | S_{k-1} = s'\}$.

Note that this particular source is characterized by long strings of zeros and long strings of ones interspersed with short transition regions. For example, starting in state 00, a representative

[16]There are two reasons why the number of decoded n-strings of source symbols by time m can be less than 2^m. The first is that the first n source symbols might not be decodable until after the mth encoded bit is received. The second is that multiple m-strings of encoded bits might lead to decoded strings with the same first n source symbols.

[17]The basic results about finite-state Markov chains, including those used here, are established in many texts such as [8] and [25] . These results are important in the further study of digital communcation, but are not essential here.

output would be

$$00000000101111111111111011111111010100000000\cdots$$

Note that if $s_k = [x_{k-1}x_k]$ then the next state must be either $s_{k+1} = [x_k0]$ or $s_{k+1} = [x_k1]$; *i.e.*, each state has only two transitions coming out of it.

The above example is now generalized to an arbitrary discrete Markov source.

Definition 2.8.1. A *finite-state Markov chain* is a sequence $S_0, S_1, \dots$ of discrete random symbols from a finite alphabet, $\mathcal{S}$. There is a pmf $q_0(s), s \in \mathcal{S}$ on S_0, and there is a conditional pmf $Q(s|s')$ such that for all $m \geq 1$, all $s \in \mathcal{S}$, and all $s' \in \mathcal{S}$,

$$\Pr(S_k{=}s|\, S_{k-1}{=}s') = \Pr(S_k{=}s|\, S_{k-1}{=}s', \dots, S_0{=}s_0) = Q(s|\, s'). \tag{2.30}$$

There is said to be a *transition* from s' to s, denoted $s' \to s$, if $Q(s|\, s') > 0$.

Note that (2.30) says, first, that the conditional probability of a state, given the past, depends only on the previous state, and second, that these transition probabilities $Q(s|s')$ do not change with time.

Definition 2.8.2. A *Markov source* is a sequence of discrete random symbols $\mathcal{X}_1, \mathcal{X}_2, \dots$ with a common alphabet $\mathcal{X}$ which is based on a finite-state Markov chain $S_0, S_1, \dots$. Each transition $(s' \to s)$ in the Markov chain is labeled with a symbol from $\mathcal{X}$; each symbol from $\mathcal{X}$ can appear on at most one outgoing transition from each state.

Note that the state alphabet $\mathcal{S}$ and the source alphabet $\mathcal{X}$ are in general different. Since each source symbol appears on at most one transition from each state, the initial state $S_0{=}s_0$, combined with the source output, $X_1{=}x_1, X_2{=}x_2, \dots$, uniquely identifies the state sequence, and, of course, the state sequence uniquely specifies the source output sequence. If $x \in \mathcal{X}$ labels the transition $s' \to s$, then the conditional probability of that x is given by $P(x|\, s') = Q(s|\, s')$. Thus, for example, in the transition $[00] \to [0]1$ in Figure 2.16, $Q([01]|\, [00]) = P(1|\, [00])$.

The reason for distinguishing the Markov chain alphabet from the source output alphabet is to allow the state to represent an arbitrary combination of past events rather than just the previous source output. It is this feature that permits Markov source models to reasonably model both simple and complex forms of memory.

A state s is *accessible* from state s' in a Markov chain if there is a path in the corresponding graph from $s' \to s$, *i.e.*, if $\Pr(S_k{=}s|\, S_0{=}s') > 0$ for some $k > 0$. The period of a state s is the greatest common divisor of the set of integers $k \geq 1$ for which $\Pr(S_k{=}s|\, S_0{=}s) > 0$. A finite-state Markov chain is *ergodic* if all states are accessible from all other states and if all states are aperiodic, *i.e.*, have period 1.

We will consider only Markov sources for which the Markov chain is ergodic. An important fact about ergodic Markov chains is that the chain has steady-state probabilities $q(s)$ for all $s \in \mathcal{S}$, given by the unique solution to the linear equations

$$\begin{aligned} q(s) &= \sum_{s' \in \mathcal{S}} q(s')Q(s|\, s'); \quad s \in \mathcal{S} \qquad (2.31)\\ \sum_{s \in \mathcal{S}} q(s) &= 1. \end{aligned}$$

These steady-state probabilities are approached asymptotically from any starting state, *i.e.*,

$$\lim_{k\to\infty} \Pr(S_k{=}s|\, S_0{=}s') = q(s) \qquad \text{for all } s, s' \in \mathcal{S}. \tag{2.32}$$

2.8.1 Coding for Markov sources

The simplest approach to coding for Markov sources is that of using a separate prefix-free code for each state in the underlying Markov chain. That is, for each $s \in \mathcal{S}$, select a prefix-free code whose lengths $l(x, s)$ are appropriate for the conditional pmf $P(x|\,s) > 0$. The codeword lengths for the code used in state s must of course satisfy the Kraft inequality $\sum_x 2^{-l(x,s)} \leq 1$. The minimum expected length, $\overline{L}_{\min}(s)$ for each such code can be generated by the Huffman algorithm and satisfies

$$\mathsf{H}[X|\,s] \leq \overline{L}_{\min}(s) < \mathsf{H}[X|\,s] + 1. \tag{2.33}$$

where, for each $s \in \mathcal{S}$, $\mathsf{H}[X|\,s] = \sum_x -P(x|\,s)\log P(x|\,s)$.

If the initial state S_0 is chosen according to the steady-state pmf $\{q(s); s \in \mathcal{S}\}$, then, from (2.31), the Markov chain remains in steady state and the overall expected codeword length is given by

$$\mathsf{H}[X|\,S] \leq \overline{L}_{\min} < \mathsf{H}[X|\,S] + 1, \tag{2.34}$$

where

$$\overline{L}_{\min} = \sum_{s\in\mathcal{S}} q(s)\overline{L}_{\min}(s) \qquad \text{and} \tag{2.35}$$

$$\mathsf{H}[X|\,S] = \sum_{s\in\mathcal{S}} q(s)\mathsf{H}[X|\,s]. \tag{2.36}$$

Assume that the encoder transmits the initial state s_0 at time 0. If M' is the number of elements in the state space, then this can be done with $\lceil \log M' \rceil$ bits, but this can be ignored since it is done only at the beginning of transmission and does not affect the long term expected number of bits per source symbol. The encoder then successively encodes each source symbol x_k using the code for the state at time $k-1$. The decoder, after decoding the initial state s_0, can decode x_1 using the code based on state s_0. After determining s_1 from s_0 and x_1, the decoder can decode x_2 using the code based on s_1. The decoder can continue decoding each source symbol, and thus the overall code is uniquely decodable. We next must understand the meaning of the conditional entropy in (2.36).

2.8.2 Conditional entropy

It turns out that the conditional entropy $\mathsf{H}[X|\,S]$ plays the same role in coding for Markov sources as the ordinary entropy $\mathsf{H}[X]$ plays for the memoryless case. Rewriting (2.36),

$$\mathsf{H}[X|\,S] = \sum_{s\in\mathcal{S}}\sum_{x\in\mathcal{X}} q(s)P(x|\,s)\log\frac{1}{P(x|\,s)}.$$

This is the expected value of the rv $\log[1/P(X|\,S)]$.

An important entropy relation, for arbitrary discrete rv's, is

$$\mathsf{H}[XS] = \mathsf{H}[S] + \mathsf{H}[X|\,S]. \tag{2.37}$$

To see this,

$$\begin{aligned}
\mathsf{H}[XS] &= \sum_{s,x} q(s)P(x|\,s)\log\frac{1}{q(s)P(x|\,s)} \\
&= \sum_{s,x} q(s)P(x|\,s)\log\frac{1}{q(s)} + \sum_{s,x} q(s)P(x|\,s)\log\frac{1}{P(x|\,s)} \\
&= \mathsf{H}[S] + \mathsf{H}[X|\,S].
\end{aligned}$$

Exercise 2.19 demonstrates that

$$\mathsf{H}[XS] \le \mathsf{H}[S] + \mathsf{H}[X]$$

Comparing this and (2.37), it follows that

$$\mathsf{H}[X|\,S] \le \mathsf{H}[X]. \tag{2.38}$$

This is an important inequality in information theory. If the entropy $\mathsf{H}[X]$ as a measure of mean uncertainty, then the conditional entropy $\mathsf{H}[X|\,S]$ should be viewed as a measure of mean uncertainty after the observation of the outcome of S. If X and S are not statistically independent, then intuition suggests that the observation of S should reduce the mean uncertainty in X; this equation indeed verifies this.

Example 2.8.2. Consider Figure 2.16 again. It is clear from symmetry that, in steady state, $p_X(0) = p_X(1) = 1/2$. Thus $\mathsf{H}[X] = 1$ bit. Conditional on S=00, X is binary with pmf $\{0.1, 0.9\}$, so $\mathsf{H}[X|\,[00]] = -0.1\log 0.1 - 0.9\log 0.9 = 0.47$ bits. Similarly, $\mathsf{H}[X|\,[11]] = 0.47$ bits, and $\mathsf{H}[X|\,[01]] = \mathsf{H}[X|\,[10]] = 1$ bit. The solution to the steady-state equations in (2.31) is $q([00]) = q([11]) = 5/12$ and $q([01]) = q([10]) = 1/12$. Thus, the conditional entropy, averaged over the states, is $\mathsf{H}[X|\,S] = 0.558$ bits.

For this example, it is particularly silly to use a different prefix-free code for the source output for each prior state. The problem is that the source is binary, and thus the prefix-free code will have length 1 for each symbol no matter what the state. As with the memoryless case, however, the use of fixed-to-variable-length codes is a solution to these problems of small alphabet sizes and integer constraints on codeword lengths.

Let $\mathsf{E}[L(\boldsymbol{X}^n)]_{\text{min,s}}$ be the minimum expected length of a prefix-free code for $\boldsymbol{X}^n$ conditional on starting in state s. Then, applying (2.13) to the situation here,

$$\mathsf{H}[\boldsymbol{X}^n \mid s] \le \mathsf{E}[L(\boldsymbol{X}^n)]_{\min,s} < \mathsf{H}[\boldsymbol{X}^n \mid s] + 1.$$

Assume as before that the Markov chain starts in steady state S_0. Thus it remains in steady state at each future time. Furthermore assume that the initial *sample state* is known at the decoder. Then the sample state continues to be known at each future time. Using a minimum expected length code for each initial sample state,

$$\mathsf{H}[\boldsymbol{X}^n \mid S_0] \le \mathsf{E}[L(\boldsymbol{X}^n)]_{\min,S_0} < \mathsf{H}[\boldsymbol{X}^n \mid S_0] + 1. \tag{2.39}$$

Since the Markov source remains in steady state, the average entropy of each source symbol given the state is $H(X \mid S_0)$, so intuition suggests (and Exercise 2.32 verifies) that

$$\mathsf{H}[\boldsymbol{X}^n \mid S_0] = n\mathsf{H}[X|\,S_0]. \tag{2.40}$$

Defining $\overline{L}_{\min,n} = \mathsf{E}[L(\boldsymbol{X}^n)]_{\min,S_0}/n$ as the minimum expected codeword length per input symbol when starting in steady state,

$$\mathsf{H}[X|\,S_0] \leq \overline{L}_{\min,n} < \mathsf{H}[X|\,S_0] + 1/n. \tag{2.41}$$

The asymptotic equipartition property (AEP) also holds for Markov sources. Here, however, there are[18] approximately $2^{n\mathsf{H}[X|\,S]}$ typical strings of length n, each with probability approximately equal to $2^{-n\mathsf{H}[X|\,S]}$. It follows as in the memoryless case that $\mathsf{H}[X|\,S]$ is the minimum possible rate at which source symbols can be encoded subject either to unique decodability or to fixed-to-fixed-length encoding with small probability of failure. The arguments are essentially the same as in the memoryless case.

The analysis of Markov sources will not be carried further here, since the additional required ideas are minor modifications of the memoryless case. Curiously, most of our insights and understanding about souce coding come from memoryless sources. At the same time, however, most sources of practical importance can be insightfully modeled as Markov and hardly any can be reasonably modeled as memoryless. In dealing with practical sources, we combine the insights from the memoryless case with modifications suggested by Markov memory.

The AEP can be generalized to a still more general class of discrete sources called *ergodic sources*. These are essentially sources for which sample time averages converge in some probabilistic sense to ensemble averages. We do not have the machinery to define ergodicity, and the additional insight that would arise from studying the AEP for this class would consist primarily of mathematical refinements.

2.9 Lempel-Ziv universal data compression

The Lempel-Ziv data compression algorithms differ from the source coding algorithms studied in previous sections in the following ways:

- They use variable-to-variable-length codes in which both the number of source symbols encoded and the number of encoded bits per codeword are variable. Moreover, the codes are time-varying.
- They do not require prior knowledge of the source statistics, yet over time they adapt so that the average codeword length $\overline{L}$ per source symbol is minimized in some sense to be discussed later. Such algorithms are called *universal*.
- They have been widely used in practice; they provide a simple approach to understanding universal data compression even though newer schemes now exist.

The Lempel-Ziv compression algorithms were developed in 1977-78. The first, LZ77 [37], uses string-matching on a sliding window; the second, LZ78 [38], uses an adaptive dictionary. The LZ78 was algorithm was implemented many years ago in UNIX `compress` and in many other places. Implementations of LZ77 appeared somewhat later (Stac Stacker, Microsoft Windows) and is still widely used.

[18]There are additional details here about whether the typical sequences include the initial state or not, but these differences become unimportant as n becomes large.

In this section, the LZ77 algorithm is described. accompanied by a high-level description of why it works. Finally, an approximate analysis of its performance on Markov sources is given, showing that it is effectively optimal.[19] In other words, although this algorithm operates in ignorance of the source statistics, it compresses substantially as well as the best algorithm designed to work with those statistics.

2.9.1 The LZ77 algorithm

The LZ77 algorithm compresses a sequence $\boldsymbol{x} = x_1, x_2, \dots$ from some given discrete alphabet $\mathcal{X}$ of size $M = |\mathcal{X}|$. At this point, no probabilistic model is assumed for the source, so $\boldsymbol{x}$ is simply a sequence of symbols, not a sequence of random symbols. A subsequence $(x_m, x_{m+1}, \dots, x_n)$ of $\boldsymbol{x}$ is represented by $\boldsymbol{x}_m^n$.

The algorithm keeps the w most recently encoded source symbols in memory. This is called a sliding window of size w. The number w is large, and can be thought of as being in the range of 2^{10} to 2^{20}, say. The parameter w is chosen to be a power of 2. Both complexity and, typically, performance increase with w.

Briefly, the algorithm operates as follows. Suppose that at some time the source symbols $\boldsymbol{x}_1^P$ have been encoded. The encoder looks for the longest match, say of length n, between the not-yet-encoded n-string $\boldsymbol{x}_{P+1}^{P+n}$ and a stored string $\boldsymbol{x}_{P+1-u}^{P+n-u}$ starting in the window of length w. The clever algorithmic idea in LZ77 is to encode this string of n symbols simply by encoding the integers n and u; *i.e.*, by pointing to the previous occurrence of this string in the sliding window. If the decoder maintains an identical window, then it can look up the string $\boldsymbol{x}_{P+1-u}^{P+n-u}$, decode it, and keep up with the encoder.

More precisely, the LZ77 algorithm operates as follows:

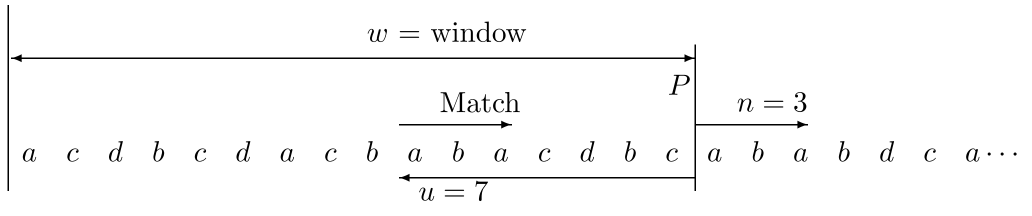
1. Encode the first w symbols in a fixed-length code without compression, using $\lceil \log M \rceil$ bits per symbol. (Since $w\lceil \log M \rceil$ will be a vanishing fraction of the total number of encoded bits, the efficiency of encoding this preamble is unimportant, at least in theory.)

2. Set the pointer $P = w$. (This indicates that all symbols up to and including x_P have been encoded.)

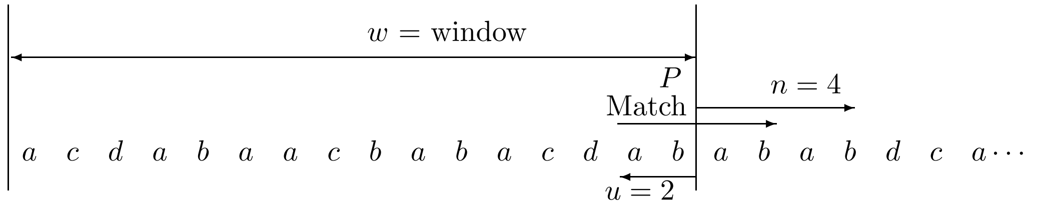
3. Find the largest $n \geq 2$ such that $\boldsymbol{x}_{P+1}^{P+n} = \boldsymbol{x}_{P+1-u}^{P+n-u}$ for some u in the range $1 \leq u \leq w$. (Find the longest match between the not-yet-encoded symbols starting at $P+1$ and a string of symbols starting in the window; let n be the length of that longest match and u the distance back into the window to the start of that match.) The string $\boldsymbol{x}_{P+1}^{P+n}$ is encoded by encoding the integers n and u.)

   Here are two examples of finding this longest match. In the first, the length of the match is $n = 3$ and the match starts $u = 7$ symbols before the pointer. In the second, the length of the match is 4 and it starts $u = 2$ symbols before the pointer. Tis illustrates that that the string and its match can overlap.

[19]A proof of this optimality for discrete ergodic sources has been given by Wyner and Ziv [36].



w = window
Match
P
$n = 3$
a c d b c d a c b a b a c d b c a b a b d c $a\cdots$
$u = 7$



w = window
P
Match
$n = 4$
a c d a b a a c b a b a c d a b a b a b d c $a\cdots$
$u = 2$

If no match exists for $n \geq 2$, then, independently of whether a match exists for $n = 1$, set $n = 1$ and directly encode the single source symbol x_{P+1} without compression.

4. Encode the integer n into a codeword from the unary-binary code. In the unary-binary code, a positive integer n is encoded into the binary representation of n, preceded by a prefix of $\lfloor \log_2 n \rfloor$ zeroes; *i.e.*,

n	prefix	base 2 expansion	codeword
1		1	1
2	0	10	010
3	0	11	011
4	00	100	00100
5	00	101	00101
6	00	110	00110
7	00	111	00111
8	000	1000	0001000

Thus the codewords starting with 0^k1 correspond to the set of 2^k integers in the range $2^k \leq n \leq 2^{k+1} - 1$. This code is prefix-free (picture the corresponding binary tree). It can be seen that the codeword for integer n has length $2\lfloor \log n \rfloor + 1$; it is seen later that this is negligible compared with the length of the encoding for u.

5. If $n > 1$, encode the positive integer $u \leq w$ using a fixed-length code of length $\log w$ bits. (At this point the decoder knows n, and can simply count back by u in the previously decoded string to find the appropriate n-tuple, even if there is overlap as above.)

6. Set the pointer P to $P + n$ and go to step (3). (Iterate forever.)

2.9.2 Why LZ77 works

The motivation behind LZ77 is information-theoretic. The underlying idea is that if the unknown source happens to be, say, a Markov source of entropy $\mathsf{H}[X|S]$, then the AEP says that, for any large n, there are roughly $2^{n\mathsf{H}[X|S]}$ typical source strings of length n. On the other hand, a window of size w contains w source strings of length n, counting duplications. This means that if $w \ll 2^{n\mathsf{H}[X|S]}$, then most typical sequences of length n cannot be found in the window, suggesting that matches of length n are unlikely. Similarly, if $w \gg 2^{n\mathsf{H}[X|S]}$, then it is reasonable to suspect that most typical sequences will be in the window, suggesting that matches of length n or more are likely.

The above argument, approximate and vague as it is, suggests that in order to achieve large typical match sizes n_t, the window w should be exponentially large, on the order of $2^{n_t\mathsf{H}[X|S]}$, which means

$$n_t \approx \frac{\log w}{\mathsf{H}[X|S]}; \qquad \text{typical match size.} \tag{2.42}$$

The encoding for a match requires $\log w$ bits for the match location and $2\lfloor \log n_t \rfloor + 1$ for the match size n_t. Since n_t is proportional to $\log w$, $\log n_t$ is negligible compared to $\log w$ for very large w. Thus, for the typical case, about $\log w$ bits are used to encode about n_t source symbols. Thus, from (2.42), the required rate, in bits per source symbol, is about $\overline{L} \approx \mathsf{H}[X|S]$.

The above argument is very imprecise, but the conclusion is that, for very large window size, $\overline{L}$ is reduced to the value required when the source is known and an optimal fixed-to-variable prefix-free code is used.

The imprecision above involves more than simply ignoring the approximation factors in the AEP. A more conceptual issue is that the strings of source symbols that must be encoded are somewhat special since they start at the end of previous matches. The other conceptual difficulty comes from ignoring the duplications of typical sequences within the window.

This argument has been made precise by Wyner and Ziv [36].

2.9.3 Discussion

Let us recapitulate the basic ideas behind the LZ77 algorithm:

1. Let N_x be the number of occurrences of symbol x in a window of very large size w. If the source satisfies the WLLN, then the relative frequency N_x/w of appearances of x in the window will satisfy $N_x/w \approx p_X(x)$ with high probability. Similarly, let $N_{\boldsymbol{x}^n}$ be the number of occurrences of $\boldsymbol{x}^n$ which start in the window. The relative frequency $N_{\boldsymbol{x}^n}/w$ will then satisfy $N_{\boldsymbol{x}^n}/w \approx p_{\boldsymbol{X}^n}(\boldsymbol{x}^n)$ with high probability for very large w. This association of relative frequencies with probabilities is what makes LZ77 a universal algorithm which needs no prior knowledge of source statistics.[20]

2. Next, as explained in the previous section, the probability of a typical source string $\boldsymbol{x}^n$ for a Markov source is approximately $2^{-n\mathsf{H}[X|S]}$. If $w >> 2^{n\mathsf{H}[X|S]}$, then, according to the previous item, $N_{\boldsymbol{x}^n} \approx w p_{\boldsymbol{X}^n}(\boldsymbol{x}^n)$ should be large and $\boldsymbol{x}^n$ should occur in the window with high probability. Alternatively, if $w << 2^{n\mathsf{H}[X|S]}$, then $\boldsymbol{x}^n$ will probably not occur. Consequently the match will usually occur for $n \approx (\log w)/\mathsf{H}[X|S]$ as w becomes very large.

3. Finally, it takes about $\log w$ bits to point to the best match in the window. The unary-binary code uses $2\lfloor \log n \rfloor + 1$ bits to encode the length n of the match. For typical n, this is on the order of $2\log(\log w/\mathsf{H}[X|S])$ which is negigible for large enough w compared to $\log w$.

[20] As Yogi Berra said, "You can observe a whole lot just by watchin'."

Consequently, LZ77 requires about $\log w$ encoded bits for each group of about $(\log w)/\mathsf{H}[X|S]$ source symbols, so it nearly achieves the optimal efficiency of $\overline{L} = \mathsf{H}[X|S]$ bits/symbol, as w becomes very large.

Discrete sources, as they appear in practice, often can be viewed over different time scales. Over very long time scales, or over the sequences presented to different physical encoders running the same algorithm, there is often very little common structure, sometimes varying from one language to another, or varying from text in a language to data from something else.

Over shorter time frames, corresponding to a single file or a single application type, there is often more structure, such as that in similar types of documents from the same language. Here it is more reasonable to view the source output as a finite length segment of, say, the output of an ergodic Markov source.

What this means is that universal data compression algorithms must be tested in practice. The fact that they behave optimally for unknown sources that can be modeled to satisfy the AEP is an important guide, but not the whole story.

The above view of different time scales also indicates that a larger window need not always improve the performance of the LZ77 algorithm. It suggests that long matches will be more likely in recent portions of the window, so that fixed length encoding of the window position is not the best approach. If shorter codewords are used for more recent matches, then it requires a shorter time for efficient coding to start to occur when the source statistics abruptly change. It also then makes sense to start coding from some arbitrary window known to both encoder and decoder rather than filling the entire window with data before starting to use the LZ77 alogorithm.

2.10 Summary of discrete source coding

Discrete source coding is important both for discrete sources such as text and computer files and also as an inner layer for discrete-time analog sequences and fully analog sources. It is essential to focus on the range of possible outputs from the source rather than any one particular output. It is also important to focus on *probabilistic* models so as to achieve the best compression for the most common outputs with less care for very rare outputs. Even universal coding techniques, such as LZ77, which are designed to work well in the absence of a probability model, require probability models to understand and evaluate how they work.

Variable-length source coding is the simplest way to provide good compression for common source outputs at the expense of rare outputs. The necessity to concatenate successive variable-length codewords leads to the non-probabilistic concept of unique decodability. Prefix-free codes provide a simple class of uniquely-decodable codes. Both prefix-free codes and the more general class of uniquely-decodable codes satisfy the Kraft inequality on the number of possible code words of each length. Moreover, for any set of lengths satisfying the Kraft inequality, there is a simple procedure for constructing a prefix-free code with those lengths. Since the expected length, and other important properties of codes, depend only on the codewords lengths (and how they are assigned to source symbols), there is usually little reason to use variable-length codes that are not also prefix free.

For a DMS with given probabilities on the symbols of a source code, the entropy is a lower bound on the expected length of uniquely decodable codes. The Huffman algorithm provides a simple procedure for finding an optimal (in the sense of minimum expected codeword length) variable-length prefix-free code. The Huffman algorithm is also useful for deriving properties about optimal variable length source codes (see Exercises 2.12 to 2.18).

All the properties of variable-length codes extend immediately to fixed-to-variable-length codes. Here the source output sequence is segmented into blocks of n symbols, each of which is then encoded as a single symbol from the alphabet of source n-tuples. For a DMS the minimum expected codeword length per source symbol then lies between $\mathsf{H}[U]$ and $\mathsf{H}[U] + 1/n$. Thus prefix-free fixed-to-variable-length codes can approach the entropy bound as closely as desired.

One of the disadvantages of fixed-to-variable-length codes is that bits leave the encoder at a variable rate relative to incoming symbols. Thus if the incoming symbols have a fixed rate and the bits must be fed into a channel at a fixed rate (perhaps with some idle periods), then the encoded bits must be queued and there is a positive probability that any finite length queue will overflow.

An alternative point of view is to consider fixed-length to fixed-length codes. Here, for a DMS, the set of possible n-tuples of symbols from the source can be partitioned into a typical set and an atypical set. For large n, the AEP says that there are essentially $2^{n\mathsf{H}[U]}$ typical n-tuples with an aggregate probability approaching 1 with increasing n. Encoding just the typical n-tuples requires about $\mathsf{H}[U]$ bits per symbol, thus approaching the entropy bound without the above queueing problem, but, of course, with occasional errors.

As detailed in the text, the AEP can be used to look at the long-term behavior of arbitrary source coding algorithms to show that the entropy bound cannot be exceeded without a failure rate that approaches 1.

The above results for discrete memoryless sources extend easily to ergodic Markov sources. The text does not carry out this analysis in detail since readers are not assumed to have the requisite knowledge about Markov chains (see [7] for the detailed analysis). The important thing here is to see that Markov sources can model n-gram statistics for any desired n and thus can model fairly general sources (at the cost of very complex models). From a practical standpoint, universal source codes, such as LZ77 are usually a more reasonable approach to complex and partly unknown sources.

2.E Exercises

2.1. Chapter 1 pointed out that voice waveforms could be converted to binary data by sampling at 8000 times per second and quantizing to 8 bits per sample, yielding 64kb/s. It then said that modern speech coders can yield telephone-quality speech at 6-16 kb/s. If your objective were simply to reproduce the words in speech recognizably without concern for speaker recognition, intonation, etc., make an estimate of how many kb/s would be required. Explain your reasoning. (Note: There is clearly no "correct answer" here; the question is too vague for that. The point of the question is to get used to questioning objectives and approaches.)

2.2. Let V and W be discrete rv's defined on some probability space with a joint pmf $p_{VW}(v, w)$.

(a) Prove that $\mathsf{E}[V + W] = \mathsf{E}[V] + \mathsf{E}[W]$. Do not assume independence.

(b) Prove that if V and W are independent rv's, then $\mathsf{E}[V \cdot W] = \mathsf{E}[V] \cdot \mathsf{E}[W]$.

c) Assume that V and W are not independent. Find an example where $\mathsf{E}[V\cdot W] \neq \mathsf{E}[V]\cdot\mathsf{E}[W]$ and another example where $\mathsf{E}[V \cdot W] = \mathsf{E}[V] \cdot \mathsf{E}[W]$.

d) Assume that V and W are independent and let σ_V^2 and σ_W^2 be the variances of V and W respectively. Show that the variance of $V + W$ is given by $\sigma_{V+W}^2 = \sigma_V^2 + \sigma_W^2$.

2.3. (a) For a nonnegative integer-valued rv N, show that $\mathsf{E}[N] = \sum_{n>0} \Pr(N \geq n)$.

(b) Show, with whatever mathematical care you feel comfortable with, that for an arbitrary nonnegative rv X that $\mathsf{E}(X) = \int_0^\infty \Pr(X \geq a)da$.

(c) Derive the Markov inequality, which says that for any $a \geq 0$ and any nonnegative rv X, $\Pr(X \geq a) \leq \frac{\mathsf{E}[X]}{a}$. Hint: Sketch $\Pr(X \geq a)$ as a function of a and compare the area of the rectangle from 0 to a on the abscissa and 0 to $\Pr(X \geq a)$ with the area corresponding to $\mathsf{E}[X]$.

(d) Derive the Chebyshev inequality, which says that $\Pr(|Y - \mathsf{E}[Y]| \geq b) \leq \frac{\sigma_Y^2}{b^2}$ for any rv Y with finite mean $\mathsf{E}[Y]$ and finite variance σ_Y^2. Hint: Use part (c) with $(Y - \mathsf{E}[Y])^2 = X$.

2.4. Let $X_1, X_2, \ldots, X_n, \ldots$ be a sequence of independent identically distributed (iid) analog rv's with the common probability density function $f_X(x)$. Note that $\Pr\{X_n{=}\alpha\} = 0$ for all α and that $\Pr\{X_n{=}X_m\} = 0$ for $m \neq n$.

(a) Find $\Pr\{X_1 \leq X_2\}$. [Give a numerical answer, not an expression; no computation is required and a one or two line explanation should be adequate.]

(b) Find $\Pr\{X_1 \leq X_2; X_1 \leq X_3\}$ (in other words, find the probability that X_1 is the smallest of $\{X_1, X_2, X_3\}$). [Again, think— don't compute.]

(c) Let the rv N be the index of the first rv in the sequence to be less than X_1; that is, $\Pr\{N{=}n\} = \Pr\{X_1 \leq X_2; X_1 \leq X_3; \cdots ; X_1 \leq X_{n-1}; X_1 > X_n\}$. Find $\Pr\{N \geq n\}$ as a function of n. Hint: generalize part (b).

(d) Show that $\mathsf{E}[N] = \infty$. Hint: use part (a) of Exercise 2.3.

(e) Now assume that $X_1, X_2 \ldots$ is a sequence of iid rv's each drawn from a finite set of values. Explain why you can't find $\Pr\{X_1 \leq X_2\}$ without knowing the pmf. Explain why $\mathsf{E}[N] = \infty$.

2.5. Let $X_1, X_2, \dots, X_n$ be a sequence of n binary iid rv's. Assume that $\Pr\{X_m{=}1\} = \Pr\{X_m{=}0\} = \frac{1}{2}$. Let Z be a parity check on $X_1, \dots, X_n$; that is, $Z = X_1 \oplus X_2 \oplus \cdots \oplus X_n$ (where $0 \oplus 0 = 1 \oplus 1 = 0$ and $0 \oplus 1 = 1 \oplus 0 = 1$).

(a) Is Z independent of X_1? (Assume $n > 1$.)

(b) Are $Z, X_1, \dots, X_{n-1}$ independent?

(c) Are $Z, X_1, \dots, X_n$ independent?

(d) Is Z independent of X_1 if $\Pr\{X_i{=}1\} \neq \frac{1}{2}$? You may take $n = 2$ here.

2.6. Define a suffix-free code as a code in which no codeword is a suffix of any other codeword.

(a) Show that suffix-free codes are uniquely decodable. Use the definition of unique decodability in Section 2.3.1, rather than the intuitive but vague idea of decodability with initial synchronization.

(b) Find an example of a suffix-free code with codeword lengths (1, 2, 2) that is not a prefix-free code. Can a codeword be decoded as soon as its last bit arrives at the decoder? Show that a decoder might have to wait for an arbitrarily long time before decoding (this is why a careful definition of unique decodability is required).

(c) Is there a code with codeword lengths (1, 2, 2) that is both prefix-free and suffix-free? Explain.

2.7. The algorithm given in essence by (2.2) for constructing prefix-free codes from a set of codeword lengths uses the assumption that the lengths have been ordered first. Give an example in which the algorithm fails if the lengths are not ordered first.

2.8. Suppose that, for some reason, you wish to encode a source into symbols from a D-ary alphabet (where D is some integer greater than 2) rather than into a binary alphabet. The development of Section 2.3 can be easily extended to the D-ary case, using D-ary trees rather than binary trees to represent prefix-free codes. Generalize the Kraft inequality, (2.1), to the D-ary case and outline why it is still valid.

2.9. Suppose a prefix-free code has symbol probabilities $p_1, p_2, \dots, p_M$ and lengths $l_1, \dots, l_M$. Suppose also that the expected length $\overline{L}$ satisfies $\overline{L} = \mathsf{H}[X]$.

(a) Explain why $p_i = 2^{-l_i}$ for each i.

(b) Explain why the sequence of encoded binary digits is a sequence of iid equiprobable binary digits. Hint: Use figure 2.4 to illustrate this phenomenon and explain in words why the result is true in general. Do not attempt a general proof.

2.10. (a) Show that in a code of M codewords satisfying the Kraft inequality with equality, the maximum length is at most $M - 1$. Explain why this ensures that the number of distinct such codes is finite.

(b) Consider the number $S(M)$ of distinct full code trees with M terminal nodes. Count two trees as being different if the corresponding set of codewords is different. That is, ignore the set of source symbols and the mapping between source symbols and codewords. Show that $S(2) = 1$ and show that for $M > 2$, $S(M) = \sum_{j=1}^{M-1} S(j)S(M-j)$ where $S(1) = 1$ by convention.

2.11. (Proof of the Kraft inequality for uniquely decodable codes) (a) Assume a uniquely decodable code has lengths $l_1, \dots, l_M$. In order to show that $\sum_j 2^{-l_j} \le 1$, demonstrate the following identity for each integer $n \ge 1$:

$$\left[\sum_{j=1}^{M} 2^{-l_j}\right]^n = \sum_{j_1=1}^{M}\sum_{j_2=1}^{M}\cdots\sum_{j_n=1}^{M} 2^{-(l_{j_1}+l_{j_2}+\cdots+l_{j_n})}$$

(b) Show that there is one term on the right for each concatenation of n codewords (*i.e.*, for the encoding of one n-tuple $\boldsymbol{x}^n$) where $l_{j_1} + l_{j_2} + \cdots + l_{j_n}$ is the aggregate length of that concatenation.

(c) Let A_i be the number of concatenations which have overall length i and show that

$$\left[\sum_{j=1}^{M} 2^{-l_j}\right]^n = \sum_{i=1}^{nl_{\max}} A_i\, 2^{-i}$$

(d) Using the unique decodability, upper bound each A_i and show that

$$\left[\sum_{j=1}^{M} 2^{-l_j}\right]^n \le nl_{\max}$$

(e) By taking the nth root and letting $n \to \infty$, demonstrate the Kraft inequality.

2.12. A source with an alphabet size of $M = |\mathcal{X}| = 4$ has symbol probabilities $\{1/3, 1/3, 2/9, 1/9\}$.

(a) Use the Huffman algorithm to find an optimal prefix-free code for this source.

(b) Use the Huffman algorithm to find another optimal prefix-free code with a different set of lengths.

(c) Find another prefix-free code that is optimal but cannot result from using the Huffman algorithm.

2.13. An alphabet of $M = 4$ symbols has probabilities $p_1 \ge p_2 \ge p_3 \ge p_4 > 0$.

(a) Show that if $p_1 = p_3 + p_4$, then a Huffman code exists with all lengths equal and another exists with a codeword of length 1, one of length 2, and two of length 3.

(b) Find the largest value of p_1, say $p_{\max}$, for which $p_1 = p_3 + p_4$ is possible.

(c) Find the smallest value of p_1, say $p_{\min}$, for which $p_1 = p_3 + p_4$ is possible.

(d) Show that if $p_1 > p_{\max}$, then every Huffman code has a length 1 codeword.

(e) Show that if $p_1 > p_{\max}$, then every optimal prefix-free code has a length 1 codeword.

(f) Show that if $p_1 < p_{\min}$, then all codewords have length 2 in every Huffman code.

(g) Suppose $M > 4$. Find the smallest value of $p'_{\max}$ such that $p_1 > p'_{\max}$ guarantees that a Huffman code will have a length 1 codeword.

2.14. Consider a source with M equiprobable symbols.

(a) Let $k = \lceil \log M \rceil$. Show that, for a Huffman code, the only possible codeword lengths are k and $k-1$.

(b) As a function of M, find how many codewords have length $k = \lceil \log M \rceil$. What is the expected codeword length $\overline{L}$ in bits per source symbol?

(c) Define $y = M/2^k$. Express $\overline{L} - \log M$ as a function of y. Find the maximum value of this function over $1/2 < y \leq 1$. This illustrates that the entropy bound, $\overline{L} < \mathsf{H}[X] + 1$ is rather loose in this equiprobable case.

2.15. Let a discrete memoryless source have M symbols with alphabet $\{1, 2, \dots, M\}$ and ordered probabilities $p_1 > p_2 > \cdots > p_M > 0$. Assume also that $p_1 < p_{M-1} + p_M$. Let $l_1, l_2, \dots, l_M$ be the lengths of a prefix-free code of minimum expected length for such a source.

(a) Show that $l_1 \leq l_2 \leq \cdots \leq l_M$.

(b) Show that if the Huffman algorithm is used to generate the above code, then $l_M \leq l_1 + 1$. Hint: Look only at the first two steps of the algorithm.

(c) Show that $l_M \leq l_1 + 1$ whether or not the Huffman algorithm is used to generate a minimum expected length prefix-free code.

(d) Suppose $M = 2^k$ for integer k. Determine $l_1, \dots, l_M$.

(e) Suppose $2^k < M < 2^{k+1}$ for integer k. Determine $l_1, \dots, l_M$.

2.16. (a) Consider extending the Huffman procedure to codes with ternary symbols $\{0, 1, 2\}$. Think in terms of codewords as leaves of ternary trees. Assume an alphabet with $M = 4$ symbols. Note that you cannot draw a full ternary tree with 4 leaves. By starting with a tree of 3 leaves and extending the tree by converting leaves into intermediate nodes, show for what values of M it is possible to have a complete ternary tree.

(b) Explain how to generalize the Huffman procedure to ternary symbols, bearing in mind your result in part (a).

(c) Use your algorithm for the set of probabilities $\{0.3, 0.2, 0.2, 0.1, 0.1, 0.1\}$.

2.17. Let X have M symbols, $\{1, 2, \dots, M\}$ with ordered probabilities $p_1 \geq p_2 \geq \cdots \geq p_M > 0$. Let X' be the reduced source after the first step of the Huffman algorithm.

(a) Express the entropy $\mathsf{H}[X]$ for the original source in terms of the entropy $\mathsf{H}[X']$ of the reduced source as

$$\mathsf{H}[X] = \mathsf{H}[X'] + (p_M + p_{M-1})H(\gamma), \tag{2.43}$$

where $H(\gamma)$ is the binary entropy function, $H(\gamma) = -\gamma \log \gamma - (1-\gamma) \log(1-\gamma)$. Find the required value of γ to satisfy (2.43).

(b) In the code tree generated by the Huffman algorithm, let v_1 denote the intermediate node that is the parent of the leaf nodes for symbols M and $M-1$. Let $q_1 = p_M + p_{M-1}$ be the probability of reaching v_1 in the code tree. Similarly, let $v_2, v_3, \dots$, denote the subsequent intermediate nodes generated by the Huffman algorithm. How many intermediate nodes are there, including the root node of the entire tree?

(c) Let $q_1, q_2, \dots$, be the probabilities of reaching the intermediate nodes $v_1, v_2, \dots$, (note that the probability of reaching the root node is 1). Show that $\overline{L} = \sum_i q_i$. Hint: Note that $\overline{L} = \overline{L}' + q_1$.

(d) Express $\mathsf{H}[X]$ as a sum over the intermediate nodes. The ith term in the sum should involve q_i and the binary entropy $H(\gamma_i)$ for some γ_i to be determined. You may find it helpful to define α_i as the probability of moving upward from intermediate node v_i, conditional on reaching v_i. (Hint: look at part a).

(e) Find the conditions (in terms of the probabilities and binary entropies above) under which $\overline{L} = \mathsf{H}[X]$.

(f) Are the formulas for $\overline{L}$ and $\mathsf{H}[X]$ above specific to Huffman codes alone, or do they apply (with the modified intermediate node probabilities and entropies) to arbitrary full prefix-free codes?

2.18. Consider a discrete random symbol X with $M+1$ symbols for which $p_1 \geq p_2 \geq \cdots \geq p_M > 0$ and $p_{M+1} = 0$. Suppose that a prefix-free code is generated for X and that for some reason, this code contains a codeword for $M+1$ (suppose for example that p_{M+1} is actaully positive but so small that it is approximated as 0).

(a) Find $\overline{L}$ for the Huffman code including symbol $M+1$ in terms of $\overline{L}$ for the Huffman code omitting a codeword for symbol $M+1$.

(b) Suppose now that instead of one symbol of zero probability, there are n such symbols. Repeat part (a) for this case.

2.19. In (2.12), it is shown that if X and Y are independent discrete random symbols, then the entropy for the random symbol XY satisfies $\mathsf{H}[XY] = \mathsf{H}[X] + \mathsf{H}[Y]$. Here we want to show that, without the assumption of independence, we have $\mathsf{H}[XY] \leq \mathsf{H}[X] + \mathsf{H}[Y]$.

(a) Show that

$$\mathsf{H}[XY] - \mathsf{H}[X] - \mathsf{H}[Y] = \sum_{x\in\mathcal{X},y\in\mathcal{Y}} p_{XY}(x,y)\log\frac{p_X(x)p_Y(y)}{p_{X,Y}(x,y)}.$$

(b) Show that $\mathsf{H}[XY] - \mathsf{H}[X] - \mathsf{H}[Y] \leq 0$, *i.e.*, that $\mathsf{H}[XY] \leq \mathsf{H}[X] + \mathsf{H}[Y]$.

(c) Let $X_1, X_2, \dots, X_n$ be discrete random symbols, not necessarily independent. Use (b) to show that

$$H[X_1X_2\cdots X_n] \leq \sum_{j=1}^{n} H[X_j].$$

2.20. Consider a random symbol X with the symbol alphabet $\{1, 2, \dots, M\}$ and a pmf $\{p_1, p_2, \dots, p_M\}$. This exercise derives a relationship called *Fano's inequality* between the entropy $\mathsf{H}[X]$ and the probability p_1 of the first symbol. This relationship is used to prove the converse to the noisy channel coding theorem. Let Y be a random symbol that is 1 if $X = 1$ and 0 otherwise. For parts (a) through (d), consider M and p_1 to be fixed.

(a) Express $\mathsf{H}[Y]$ in terms of the binary entropy function, $H_b(\alpha) = -\alpha\log(\alpha) - (1-\alpha)\log(1-\alpha)$.

(b) What is the conditional entropy $\mathsf{H}[X \mid Y{=}1]$?

(c) Show that $\mathsf{H}[X \mid Y{=}0] \leq \log(M-1)$ and show how this bound can be met with equality by appropriate choice of $p_2, \dots, p_M$. Combine this with part (c) to upper bound $\mathsf{H}[X|Y]$.

(d) Find the relationship between $\mathsf{H}[X]$ and $\mathsf{H}[XY]$

(e) Use $\mathsf{H}[Y]$ and $\mathsf{H}[X|Y]$ to upper bound $\mathsf{H}[X]$ and show that the bound can be met with equality by appropriate choice of $p_2, \dots, p_M$.

(f) For the same value of M as before, let $p_1, \dots, p_M$ be arbitrary and let $p_{\max}$ be $\max\{p_1, \dots, p_M\}$. Is your upper bound in (d) still valid if you replace p_1 by $p_{\max}$? Explain.

2.21. A discrete memoryless source emits iid random symbols $X_1, X_2, \dots$. Each random symbol X has the symbols $\{a, b, c\}$ with probabilities $\{0.5, 0.4, 0.1\}$, respectively.

(a) Find the expected length $\overline{L}_{\min}$ of the best variable-length prefix-free code for X.

(b) Find the expected length $\overline{L}_{\min,2}$, normalized to bits per symbol, of the best variable-length prefix-free code for X^2.

(c) Is it true that for any DMS, $\overline{L}_{\min} \geq \overline{L}_{\min,2}$? Explain.

2.22. For a DMS X with alphabet $\mathcal{X} = \{1, 2, \dots, M\}$, let $L_{\min,1}$, $L_{\min,2}$, and $L_{\min,3}$ be the normalized average lengths, in bits per source symbol, for a Huffman code over $\mathcal{X}$, $\mathcal{X}^2$ and $\mathcal{X}^3$ respectively. Show that $L_{\min,3} \leq \frac{2}{3}L_{\min,2} + \frac{1}{3}L_{\min,1}$.

2.23. (Run-Length Coding) Suppose $X_1, X_2, \dots$, is a sequence of binary random symbols with $p_X(a) = 0.9$ and $p_X(b) = 0.1$. We encode this source by a variable-to-variable-length encoding technique known as run-length coding. The source output is first mapped into intermediate digits by counting the number of occurences of a between each successive b. Thus an intermediate output occurs on each occurence of the symbol b. Since we don't want the intermediate digits to get too large, however, the intermediate digit 8 is used on the eighth consecutive a and the counting restarts at this point. Thus, outputs appear on each b and on each 8 a's. For example, the first two lines below illustrate a string of source outputs and the corresponding intermediate outputs.

b	a	a	a	b	a	a	a	a	a	a	a	a	a	a	b	b	a	a	a	a	b
0				3								8			2	0					4
0000				0011								1			0010	0000					0100

The final stage of encoding assigns the codeword 1 to the intermediate integer 8, and assigns a 4 bit codeword consisting of 0 followed by the three bit binary representation for each integer 0 to 7. This is illustrated in the third line above.

(a) Show why the overall code is uniquely decodable.

(b) Find the expected total number of output bits corresponding to each occurrence of the letter b. This total number includes the four bit encoding of the letter b *and* the one bit encoding for each consecutive string of 8 letter a's preceding that letter b.

(c) By considering a string of 10^{20} binary symbols into the encoder, show that the number of b's to occur per input symbol is, with very high probability, very close to 0.1.

(d) Combine parts (b) and (c) to find the $\overline{L}$, the expected number of output bits per input symbol.

2.24. (a) Suppose a DMS emits h and t with probability 1/2 each. For $\varepsilon = 0.01$, what is T_ε^5?

(b) Find T_ε^1 for $\Pr(h) = 0.1$, $\Pr(t) = 0.9$, and $\varepsilon = 0.001$.

2.25. Consider a DMS with a two symbol alphabet, $\{a, b\}$ where $p_X(a) = 2/3$ and $p_X(b) = 1/3$. Let $\boldsymbol{X}^n = X_1, \dots, X_n$ be a string of random symbols from the source with $n = 100,000$.

(a) Let $W(X_j)$ be the log pmf rv for the jth source output, *i.e.*, $W(X_j) = -\log 2/3$ for $X_j = a$ and $-\log 1/3$ for $X_j = b$. Find the variance of $W(X_j)$.

(b) For $\varepsilon = 0.01$, evaluate the bound on the probability of the typical set given in (2.24).

(c) Let N_a be the number of a's in the string $\boldsymbol{X}^n = X_1, \dots, X_n$. The rv N_a is the sum of n iid rv's. Show what these rv's are.

(d) Express the rv $W(\boldsymbol{X}^n)$ as a function of the rv N_a. Note how this depends on n.

(e) Express the typical set in terms of bounds on N_a (*i.e.*, $T_\varepsilon^n = \{\boldsymbol{x}^n : \alpha < N_a < \beta\}$ and calculate α and β).

(f) Find the mean and variance of N_a. Approximate $\Pr\{T_\varepsilon^n\}$ by the central limit theorem approximation. The central limit theorem approximation is to evaluate $\Pr\{T_\varepsilon^n\}$ assuming that N_a is Gaussian with the mean and variance of the actual N_a.

One point of this exercise is to illustrate that the Chebyshev inequality used in bounding $\Pr(T_\varepsilon)$ in the text is very weak (although it is a strict bound, whereas the Gaussian approximation here is relatively accurate but not a bound). Another point is to show that n must be very large for the typical set to look typical.

2.26. For the rv's in the previous exercise, find $\Pr\{N_a = i\}$ for $i = 0, 1, 2$. Find the probability of each individual string $\boldsymbol{x}^n$ for those values of i. Find the particular string $\boldsymbol{x}^n$ that has maximum probability over all sample values of $\boldsymbol{X}^n$. What are the next most probable n-strings? Give a brief discussion of why the most probable n-strings are not regarded as typical strings.

2.27. Let $X_1, X_2, \dots,$ be a sequence of iid symbols from a finite alphabet. For any block length n and any small number $\varepsilon > 0$, define the *good* set of n-tuples $\mathbf{x^n}$ as the set

$$G_\varepsilon^n = \left\{ \mathbf{x}^n : \; p_{\mathbf{X}^n}(\mathbf{x}^n) > 2^{-n[\mathsf{H}[X]+\varepsilon]} \right\}.$$

(a) Explain how G_ε^n differs from the typical set T_ε^n.

(b) Show that $\Pr(G_\varepsilon^n) \geq 1 - \frac{\sigma_W^2}{n\varepsilon^2}$ where W is the log pmf rv for X. Nothing elaborate is expected here.

(c) Derive an upper bound on the number of elements in G_ε^n of the form $|G_\varepsilon^n| < 2^{n(\mathsf{H}[X]+\alpha)}$ and determine the value of α. (You are expected to find the smallest such α that you can, but not to prove that no smaller value can be used in an upper bound).

(d) Let $G_\varepsilon^n - T_\varepsilon^n$ be the set of n-tuples $\boldsymbol{x}^n$ that lie in G_ε^n but not in T_ε^n. Find an upper bound to $|G_\varepsilon^n - T_\varepsilon^n|$ of the form $|G_\varepsilon^n - T_\varepsilon^n| \leq 2^{n(\mathsf{H}[X]+\beta)}$. Again find the smallest β that you can.

(e) Find the limit of $|G_\varepsilon^n - T_\varepsilon^n|/|T_\varepsilon^n|$ as $n \to \infty$.

2.28. The typical set T_ε^n defined in the text is often called a weakly typical set, in contrast to another kind of typical set called a strongly typical set. Assume a discrete memoryless source and let $N_j(\boldsymbol{x}^n)$ be the number of symbols in an n string $\boldsymbol{x}^n$ taking on the value j. Then the strongly typical set S_ε^n is defined as

$$S_\varepsilon^n = \left\{ \boldsymbol{x}^n : \; p_j(1-\varepsilon) < \frac{N_j(\boldsymbol{x}^n)}{n} < p_j(1+\varepsilon); \quad \text{for all}\, j \in \mathcal{X} \right\}.$$

(a) Show that $p_{\boldsymbol{X}^n}(\boldsymbol{x}^n) = \prod_j p_j^{N_j(\boldsymbol{x}^n)}$.

(b) Show that every $\boldsymbol{x}^n$ in S_ε^n has the property that

$$\mathsf{H}[X](1-\varepsilon) < \frac{-\log p_{\boldsymbol{X}^n}(\boldsymbol{x}^n)}{n} < \mathsf{H}[X](1+\varepsilon)$$

(c) Show that if $\boldsymbol{x}^n \in S_\varepsilon^n$, then $\boldsymbol{x}^n \in T_{\varepsilon'}^n$ with $\varepsilon' = \mathsf{H}[X]\varepsilon$, *i.e.*, that $S_\varepsilon^n \subseteq T_{\varepsilon'}^n$.

(d) Show that for any $\delta > 0$ and all sufficiently large n,

$$\Pr\left(\boldsymbol{X}^n \notin S_\varepsilon^n\right) \leq \delta$$

Hint:Taking each letter j separately, $1 \leq j \leq M$, show that for all sufficiently large n, $\Pr\left(\left|\frac{N_j}{n} - p_j\right| \geq \varepsilon\right) \leq \frac{\delta}{M}$.

(e) Show that for all $\delta > 0$ and all suffiently large n,

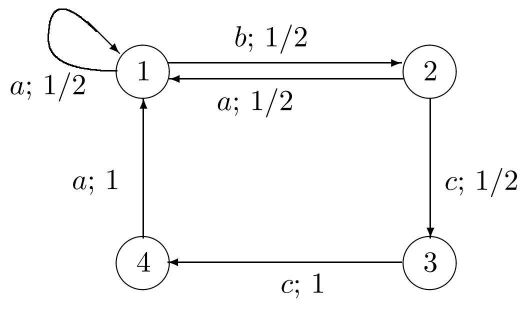
$$(1-\delta)2^{n(\mathsf{H}[X]-\varepsilon)} < |S_\varepsilon^n| < 2^{n(\mathsf{H}[X]+\varepsilon)}. \tag{2.44}$$

Note that parts (d) and (e) constitute the same theorem for the strongly typical set as Theorem 2.7.1 establishes for the weakly typical set. Typically the n required for (2.44) to hold (with the above correspondence between ε and ε) is considerably larger than than that for (2.27) to hold. We will use strong typicality later in proving the noisy channel coding theorem.

2.29. (a) The random variable D_n in Subsection 2.7.4 was defined as the initial string length of encoded bits required to decode the first n symbols of the source input. For the run-length coding example in Exercise 2.23, list the input strings and corresponding encoded output strings that must be inspected to decode the first source letter and from this find the pmf function of D_1. Hint: As many as 8 source letters must be encoded before X_1 can be decoded.

(b)Find the pmf of D_2. One point of this exercise is to convince you that D_n is a useful rv for proving theorems, but not a rv that is useful for detailed computation. It also shows clearly that D_n can depend on more than the first n source letters.

2.30. The Markov chain $S_0, S_1, \ldots$ below starts in steady state at time 0 and has 4 states, $\mathcal{S} = \{1, 2, 3, 4\}$. The corresponding Markov source $X_1, X_2, \ldots$ has a source alphabet $\mathcal{X} = \{a, b, c\}$ of size 3.



(a) Find the steady-state probabilities $\{q(s)\}$ of the Markov chain.

(b) Find $\mathsf{H}[X_1]$.

(c) Find $\mathsf{H}[X_1|S_0]$.

(d) Describe a uniquely-decodable encoder for which $\overline{L} = \mathsf{H}[X_1|S_0)$. Assume that the initial state is known to the decoder. Explain why the decoder can track the state after time 0.

(e) Suppose you observe the source output without knowing the state. What is the maximum number of source symbols you must observe before knowing the state?

2.31. Let $X_1, X_2, \ldots, X_n$ be discrete random symbols. Derive the following chain rule:

$$\mathsf{H}[X_1, \ldots, X_n] = \mathsf{H}[X_1] + \sum_{k=2}^{n} \mathsf{H}[X_k|X_1, \ldots, X_{k-1}]$$

Hint: Use the chain rule for $n = 2$ in (2.37) and ask yourself whether a k tuple of random symbols is itself a random symbol.

2.32. Consider a discrete ergodic Markov chain $S_0, S_1, \ldots$ with an arbitrary initial state distribution.

(a) Show that $\mathsf{H}[S_2|S_1S_0] = \mathsf{H}[S_2|S_1]$ (use the basic definition of conditional entropy).

(b) Show with the help of Exercise 2.31 that for any $n \geq 2$,

$$\mathsf{H}[S_1S_2 \cdots S_n|S_0] = \sum_{k=1}^{n} \mathsf{H}[S_k|S_{k-1}].$$

(c) Simplify this for the case where S_0 is in steady state.

(d) For a Markov source with outputs $X_1X_2\cdots$, explain why $\mathsf{H}[X_1 \cdots X_n|S_0] = \mathsf{H}[S_1 \cdots S_n|S_0]$. You may restrict this to $n = 2$ if you desire.

(e) Verify (2.40).

2.33. Perform an LZ77 parsing of the string $\underline{00011101}0010101100$. Assume a window of length $W = 8$; the initial window is underlined above. You should parse the rest of the string using the Lempel-Ziv algorithm.

2.34. Suppose that the LZ77 algorithm is used on the binary string $x_1^{10,000} = 0^{5000}1^{4000}0^{1000}$. This notation means 5000 repetitions of 0 followed by 4000 repetitions of 1 followed by 1000 repetitions of 0. Assume a window size $w = 1024$.

(a) Describe how the above string would be encoded. Give the encoded string and describe its substrings.

(b) How long is the encoded string?

(c) Suppose that the window size is reduced to $w = 8$. How long would the encoded string be in this case? (Note that such a small window size would only work well for really simple examples like this one.)

(d) Create a Markov source model with 2 states that is a reasonably good model for this source output. You are not expected to do anything very elaborate here; just use common sense.

(e) Find the entropy in bits per source symbol for your source model.

2.35. (a) Show that if an optimum (in the sense of minimum expected length) prefix-free code is chosen for any given pmf (subject to the condition $p_i > p_j$ for $i < j$), the code word lengths satisfy $l_i \leq l_j$ for all $i < j$. Use this to show that for all $j \geq 1$

$$l_j \geq \lfloor \log j \rfloor + 1$$

(b) The asymptotic efficiency of a prefix-free code for the positive integers is defined to be $\lim_{j\to\infty} \frac{l_j}{\log j}$. What is the asymptotic efficiency of the unary-binary code?

(c) Explain how to construct a prefix-free code for the positive integers where the asymptotic efficiency is 1. Hint: Replace the unary code for the integers $n = \lfloor \log j \rfloor + 1$ in the unary-binary code with a code whose length grows more slowly with increasing n.