6. (a) Show that the series $\sum \frac{1}{n \log^2 n}$ converges. Denote the sum by S.

   (b) Let $p_n = \frac{1}{Sn \log^2 n}$. Is $H(p_1,p_2,\ldots)$ finite or infinite?

7. (a) Show that the series $\sum \frac{1}{n^k}$ converges for $k \geq 2$. Denote the sum by S.

   (b) Let $p_n = \frac{1}{Sn^k}$. Show that $H(p_1,p_2,\ldots)$ is finite.

8. Let $S_1 = \{0,1,2\}$ be a source, with $P(0) = p$, $P(1) = q$, $P(2) = 1 - p - q$. Repeatedly performing the experiment of sampling this source until a 2 appears produces another source with alphabet $S_2 = \{a_1 \cdots a_k 2 \mid a_i \in S_1,\ a_i \neq 2\}$. Calculate the probability distribution and the entropy of this source.

9. Let $\{p_1,p_2,\ldots\}$ be a countably infinite probability distribution, whose entropy is finite. Can you approximate the entropy $H(p_1,p_2,\ldots)$ to any desired degree of accuracy by the entropy of a finite probability distribution? Explain.

10. Show by example that the requirement of being monotonically decreasing is essential to part 2) of Theorem 1.3.3. In other words, find an example of a probability distribution $\{p_1,p_2,\ldots\}$ for which the sequence $p_1,p_2,\ldots$ is not monotonically decreasing and for which $\sum p_i \log \frac{1}{p_i}$ converges, but $\sum p_i \log i$ does not converge.

CHAPTER 2

Noiseless Coding

2.1 Variable Length Encoding

Now we turn to a discussion of *source encoding for noiseless transmission.* When no errors can occur in the transmission of data, we may concentrate on the question of how to encode the data as efficiently as possible, in a sense we will make precise in a moment. First, let us set some basic terminology.

STRINGS AND CODES

Let $\mathcal{A} = \{a_1,\ldots,a_n\}$ be a finite set, which we refer to as an **alphabet.** A **string,** or **word,** over the alphabet $\mathcal{A}$ is any sequence of elements of $\mathcal{A}$. We will usually (but not always) write strings in the form

$$\mathbf{a} = a_{i_1} a_{i_2} \cdots a_{i_k}$$

using juxtaposition of symbols. Occasionally, for readability sake, we may include spaces, commas, parentheses, or other punctuation marks, between the symbols in a string. The **empty string** θ is the unique string with no symbols.

The **length** of a string $\mathbf{a}$, denoted by $len(\mathbf{a})$, is the number of alphabet symbols appearing in the string. The set of all strings over $\mathcal{A}$ will be denoted by $\mathcal{A}^*$.

Definition Let $\mathcal{A} = \{a_1,\ldots,a_r\}$ be a finite set, which we call a **code alphabet.** An **r-ary code** is a nonempty subset C of the set $\mathcal{A}^*$ of all

strings over $\mathcal{A}$. The size r of the code alphabet is called the **radix** of the code, and the elements of the code are called **codewords.** A code whose alphabet is {0,1} is called a **binary code**, and a code whose alphabet is {0,1,2} is called a **ternary code.** □

Definition Let $\mathcal{S} = (S,P)$ be a source. An **encoding scheme** for $\mathcal{S}$ is an ordered pair (C,f), where C is a code and $f:S \to C$ is an *injective* function, called an **encoding function.** □

Thus, an encoding function assigns a codeword from C to each source symbol in S.

AVERAGE CODEWORD LENGTH

For the purposes of noiseless encoding, the measure of efficiency of an encoding scheme is its *average codeword length.*

Definition The **average codeword length** of an encoding scheme (C,f) for a source $\mathcal{S} = (S,P)$, where $S = \{s_1,\ldots,s_n\}$, is defined by

$$AveLen(C,f) = \sum_{i=1}^{n} P(s_i)len(f(s_i))$$ □

Example 2.1.1 Consider the source $S = \{a,b,c,d\}$, with probabilities $P(a) = P(b) = 2/17$, $P(c) = 9/17$ and $P(d) = 4/17$. Consider also the encoding schemes (C_1,f_1) and (C_2,f_2), where

$$C_1 = \{0,11,100,101\} \qquad C_2 = \{00,10,11,01010\}$$

$$\begin{array}{ll} f_1(a) = 11 & f_2(a) = 01010 \\ f_1(b) = 0 & f_2(b) = 00 \\ f_1(c) = 100 & f_2(c) = 10 \\ f_1(d) = 10 & f_2(d) = 11 \end{array}$$

Since

$$AveLen(C_1,f_1) = \frac{2}{17}\cdot 2 + \frac{2}{17}\cdot 1 + \frac{9}{17}\cdot 3 + \frac{4}{17}\cdot 2 = \frac{41}{17}$$

and

$$AveLen(C_2,f_2) = \frac{2}{17}\cdot 5 + \frac{2}{17}\cdot 2 + \frac{9}{17}\cdot 2 + \frac{4}{17}\cdot 2 = \frac{40}{17}$$

we see that (C_2,f_2) has a smaller average codeword length, and so is more efficient than (C_1,f_1), even though *the code* C_2 has longer codewords (on the average) than *the code* C_1. This emphasizes the fact that the average codeword length of an *encoding scheme* is not the same as the average codeword length of a *code*, since the former depends also on the probability distribution P. □

We should point out that it makes sense to compare the average codeword lengths of different encoding schemes only when the corresponding codes have the same radix. For in general, the larger the radix, the shorter we can make the average codeword length.

Our goal in this chapter is to determine the *minimum* average codeword length among all "good" encoding schemes (in a sense we will make precise soon), as well as to find a method for constructing such encoding schemes. As we will see, both goals are readily achieved.

FIXED AND VARIABLE LENGTH CODES

Definition If all the codewords in a code C have the same length, we say that C is a **fixed length code**, or **block code.** If C contains codewords of different lengths, we say that C is a **variable length code.**

Any encoding scheme that uses a fixed length code will be referred to as a **fixed length encoding scheme**, and similarly for **variable length encoding schemes.** □

When the probability distribution P is not uniform, variable length encoding is usually more efficient than fixed length encoding. As a simple example, consider a source with alphabet $S = \{s_1,\ldots,s_5\}$, whose probability distribution satisfies

$$P(s_1) = 1-\epsilon \quad \text{and} \quad P(\{s_2,s_3,s_4,s_5\}) = \epsilon$$

Since a fixed length binary code must have codeword length at least 3, in order to encode 5 words, its average codeword length is also at least 3. On the other hand, using a variable length code, we may assign the codeword 0 to s_1 and the codewords 100, 101, 110, and 111 to the other source symbols, giving an average codeword length of $1\cdot(1-\epsilon)+3\epsilon = 1+2\epsilon$, which is less than 3 if $\epsilon < 1$.

UNIQUE DECIPHERABILITY

Even though variable length encoding schemes can be more efficient than fixed length schemes, there is a potential problem with variable length schemes, as illustrated by the following example.

$$S = \{a,b,c\}, \quad C = \{0,01,001\}$$
$$f(a) = 0, \quad f(b) = 01, \quad f(c) = 001$$

This encoding scheme is not *uniquely decipherable*, in the sense that the codeword string 001 could be decoded as *ab* or as *c*. In order to make this encoding scheme uniquely decipherable, we require a

codeword separator, such as /, which enables us to write the message *ab* as 0/01. Of course, the addition of a codeword separator adds to the overall length of encoded messages, which is contrary to the goal of efficient encoding. (Fixed length encoding schemes are automatically uniquely decipherable and need no codeword separator.)

The difficulty here can be traced to the fact that a string of code alphabet symbols may represent more than one string of codewords. This leads to the following definition.

Definition A code C is **uniquely decipherable** if whenever $\mathbf{c}_1,\ldots,\mathbf{c}_k$, $\mathbf{d}_1,\ldots,\mathbf{d}_j$ are codewords in C and

$$\mathbf{c}_1\cdots\mathbf{c}_k = \mathbf{d}_1\cdots\mathbf{d}_j$$

then $k = j$ and $\mathbf{c}_i = \mathbf{d}_i$ for all $i = 1,\ldots,k$. □

Clearly, the property of being uniquely decipherable is extremely desirable. Surprisingly, even a small change can make a code that is not uniquely decipherable into one that is.

Example 2.1.2 Let $S = \{a,b,c\}$ and consider the encoding scheme

$$C = \{1,01,001\}$$
$$f(a) = 1, \quad f(b) = 01, \quad f(c) = 001$$

This differs from the previous code only in that the codeword 0 is replaced by the codeword 1. However, this code is uniquely decipherable. To see this, observe that the symbol 1 acts as a kind of codeword separator, in the sense that the presence of a 1 indicates the end of a codeword. Thus, reading a codeword string from left to right, we *must* decode *when and only when* we encounter a 1. For instance, consider the string 1001011. Reading from left to right, we *must* decode 1 as *a*, 001 as *c*, 01 as *b*, and 1 as *a* to get the source string *acba*. No other decoding is possible. □

Although there are methods for showing that a particular code is uniquely decipherable, we shall not go into them here, since we will limit our discussion to a special type of uniquely decipherable code, without limiting our ability to be efficient.

To be more specific, one of the difficulties with unique decipherability is that, even though a code may have this property, it may be necessary to wait until the entire message has been received before we can begin to decode.

Example 2.1.3 Consider the code $C = \{0,01,001\}$ and the encoding function

$$f(a) = 0, \quad f(b) = 01, \quad f(c) = 011, \quad f(d) = 0111$$

It is not hard to see that this code is uniquely decipherable. Now suppose that the string 0111 is being transmitted. Just after receiving the first 0, we cannot tell whether it represents the source letter *a*, or the beginning of a different source letter. Similarly, when the first 01 is received, we cannot tell whether it represents a *b*, or the beginning of a *c* or *d*. In fact, we cannot decipher the source message 0111 until it has been completely received.

On the other hand, consider the code $D = \{0,10,110,1110\}$ and encoding function

$$g(a) = 0, \quad g(b) = 10, \quad g(c) = 110, \quad g(d) = 1110$$

In this case, individual codewords can be deciphered as soon as they are received, since the presence of a 0 indicates the end of a codeword. Thus, each source symbol can be decoded as soon as its codeword is received. □

INSTANTANEOUS CODES; THE PREFIX PROPERTY

The previous example prompts us to make the following definition.

Definition A code is said to be **instantaneous** if each codeword in any string of codewords can be decoded (reading from left to right) as soon as it is received. □

If a code is instantaneous, then it is also uniquely decipherable. However, as the code C of Example 2.1.3 illustrates, the converse is not true.

The property of being instantaneous is very desirable. Fortunately, there is a very simple way to tell when a code has this property. First we need a definition.

Definition A code is said to have the **prefix property** if no codeword is a prefix of any other codeword, that is, if whenever $\mathbf{c} = x_1x_2\cdots x_n$ is a codeword, then $x_1x_2\cdots x_k$ is not a codeword for $1 \le k < n$. □

Given a code C, it is a simple matter to determine whether or not it has the prefix property. It is only necessary to compare each codeword with all codewords of equal or greater length to see if it is a prefix. For example, the code {1,01,001} has the prefix property, since 1 is not a prefix of 01 or 001 and 01 is not a prefix of 001. However, the code {0,01,001} does not have the prefix property, since

0 is a prefix of 01.

The importance of the prefix property comes from the following theorem, whose proof we leave as an exercise.

Theorem 2.1.1 A code C is instantaneous if and only if it has the prefix property. □

Example 2.1.4 Let n be a positive integer. A **comma code** is a code C with codewords

$$1,\ 01,\ 001,\ 0001,\ldots,\ \underbrace{0\cdots0}_{n-1}1,\ \underbrace{0\cdots0}_{n}$$

This terminology comes from the fact that the symbol 1 acts as a kind of comma, indicating the end of a codeword. (The last codeword is determined by the unique number of 0's that it contains.) Since comma codes have the prefix property, they are instantaneous. On the other hand, the code

$$1,\ 10,\ 100,\ 1000,\ldots,\ 1\underbrace{0\cdots0}_{n-1},\ \underbrace{0\cdots0}_{n}$$

does not have the prefix property, and so it is not instantaneous. However, it is uniquely decipherable, since we can decipher any string of codewords by reading from right to left, where a 1 indicates the *beginning* of a codeword. □

KRAFT'S THEOREM

The following remarkable theorem, published by L.G. Kraft in 1949, gives a simple criterion to determine whether or not there is an instantaneous code with given codeword lengths.

Theorem 2.1.2 (Kraft's Theorem)

1) If C is an r-ary instantaneous code with codeword lengths $\ell_1,\ldots,\ell_n$, then these lengths must satisfy **Kraft's inequality**

$$\sum_{k=1}^{n} \frac{1}{r^{\ell_k}} \le 1$$

2) If the numbers $\ell_1,\ \ell_2,\ \ldots,\ \ell_n$ and r satisfy **Kraft's inequality**, then there is an instantaneous r-ary code with codeword lengths $\ell_1,\ldots,\ell_n$.

Proof. Suppose first that $C = \{\mathbf{c}_1,\ldots,\mathbf{c}_n\}$ is an instantaneous r-ary code with codeword lengths $\ell_1,\ldots,\ell_n$. We will show that Kraft's inequality must hold. Let $L = \max\{\ell_i\}$. If $\mathbf{c}_i = x_1x_2\cdots x_{\ell_i} \in C$, then

any word of the form

$$\mathbf{x} = x_1x_2\cdots x_{\ell_i}y_{\ell_i+1}\cdots y_L \tag{2.1.1}$$

where the y_j are any code symbols, cannot be in C, because $\mathbf{c}_i$ is a prefix of $\mathbf{x}$. But there are a total of $r^{L-\ell_i}$ words of the form (2.1.1). Summing on i, we see that there are

$$\sum_{i=1}^{n} r^{L-\ell_i} = r^L \sum_{i=1}^{n} \frac{1}{r^{\ell_i}}$$

words of length L that cannot be in C. However, the total number of words of length L over the code alphabet is r^L, and so we must have

$$r^L \sum_{i=1}^{n} \frac{1}{r^{\ell_i}} \le r^L$$

or

$$\sum_{i=1}^{n} \frac{1}{r^{\ell_i}} \le 1$$

which is Kraft's inequality.

Now suppose that $\ell_1,\ \ell_2,\ldots,\ell_n$ and r satisfy Kraft's inequality. We will show that there exists an instantaneous code C, over an alphabet $\mathcal{A} = \{a_1,\ a_2,\ \ldots,\ a_r\}$, with codeword lengths ℓ_i. Let α_j be the number of ℓ_i that are equal to j. Thus, α_1 is the number of desired codewords of length 1, α_2 is the number of desired codewords of length 2, and so on.

In order to construct the desired code, we want to select α_1 words of length 1, say the first α_1 code letters

$$a_1,\ a_2,\ \ldots,\ a_{\alpha_1} \tag{2.1.2}$$

This can be done as long as

$$\alpha_1 \le r$$

Next, we want to select α_2 words of length 2. However, since our code must be instantaneous, we cannot allow any of the α_1 codewords in (2.1.2) to be prefixes of the new codewords. In other words, from among the r^2 possible words of length 2 over $\mathcal{A}$, we cannot select the $\alpha_1 r$ codewords that begin with any of the α_1 codewords in (2.1.2). This leaves $r^2 - \alpha_1 r$ codewords from which to choose α_2 codewords, and this can be done provided that

$$\alpha_2 \le r^2 - \alpha_1 r$$

or

$$\alpha_1 r + \alpha_2 \le r^2$$