Steven Roman

Introduction to Coding and Information Theory

With 50 illustrations

Springer

1 An Introduction to Codes

CHAPTER

1.1 Strings and Things

Modular Arithmetic in $\mathbb{Z}_n$

If n is a positive integer, we let $\mathbb{Z}_n$ denote the set $\{0, 1, \ldots, n-1\}$ consisting of the first n non-negative integers. Much of what we will do in this book (especially in the coding theory portion) involves this set.

If α and β are integers, the following three conditions are equivalent.

1. $\alpha - \beta$ is divisible by n
2. there exists an integer k for which $\alpha = \beta + kn$
3. α and β have the same remainder when divided by n.

If any (and hence all) of these conditions hold, we say that α and β are **congruent modulo** n, and write

$$\alpha \equiv \beta (\mathrm{mod} n)$$

Example 1.1.1

1. $5 \equiv 3(\mathrm{mod}2)$
2. $178 \equiv 17(\mathrm{mod}7)$
3. $-4 \equiv 12(\mathrm{mod}4)$

4. $15 \equiv 0 (\mathrm{mod} 5)$
5. $4 \not\equiv 6 (\mathrm{mod} 3)$
6. $14 \equiv 9 \equiv 4 \equiv -1 \equiv -6 (\mathrm{mod} 5)$ □

Given an integer α, there is exactly one integer k in $\mathbb{Z}_n$ that is congruent to α, that is, for which $k \equiv \alpha (\mathrm{mod} n)$. Put another way, in $\mathbb{Z}_n$, there exists a unique solution to the equation

$$x \equiv \alpha (\mathrm{mod} n)$$

This solution is simply the remainder obtained by dividing α by n and is referred to as the **residue** of α modulo n. It can also be found by adding, or subtracting, a suitable multiple of n in order to produce a number in $\mathbb{Z}_n$. For instance, the residue of 25 modulo 7 is found by subtracting $3 \cdot 7 = 21$ from 25, to get 4. That is, $4 \equiv 25 \pmod 7$, where $4 \in \mathbb{Z}_7$.

We can define two algebraic operations, known as **addition modulo** n and **multiplication modulo** n, on the set $\mathbb{Z}_n$. If $x, y \in \mathbb{Z}_n$, we set

$$x \oplus_n y = \text{the remainder obtained by dividing } x + y \text{ by } n$$

and

$$x \otimes_n y = \text{the remainder obtained by dividing } xy \text{ by } n$$

Hence, $x \oplus_n y \equiv (x + y) \pmod n$ and $x \otimes_n y \equiv xy \pmod n$.

Example 1.1.2

1. In $\mathbb{Z}_5$,

$$3 \oplus_5 4 = 2 \text{ and } 3 \otimes_5 2 = 1$$

2. In $\mathbb{Z}_{10}$,

$$7 \oplus_{10} 8 = 5 \text{ and } 7 \otimes_{10} 8 = 6$$ □

The set $\mathbb{Z}_n$, together with the operations of addition and multiplication modulo n, is referred to as the **integers modulo** n. It is customary, whenever the context makes it clear, to use the ordinary symbols $+$ and $\cdot$, instead of $\oplus_n$ and $\otimes_n$, and we will also follow this custom.

In $\mathbb{Z}_2$, addition and multiplication modulo 2 are described by the following tables.

Addition modulo 2		
+	0	1
0	0	1
1	1	0

Multiplication modulo 2		
·	0	1
0	0	0
1	0	1

The Field $\mathbb{Z}_p$

Our main interest in the set $\mathbb{Z}_p$ is when p is a prime. The reason is that, when p is a prime, addition and multiplication modulo p have much nicer properties than in the nonprime case. The following theorem is a case in point.

Theorem 1.1.1 *The following property holds in $\mathbb{Z}_n$ if and only if n is a prime*

$$\alpha\beta = 0 \text{ implies } \alpha = 0 \text{ or } \beta = 0$$

Proof Let $n = p$ be a prime and suppose that $\alpha\beta = 0$ in $\mathbb{Z}_p$. This is equivalent to $\alpha\beta \equiv 0 \pmod{p}$, which holds if and only if p divides $\alpha\beta$. Since p is a prime, it divides the product $\alpha\beta$ if and only if it divides at least one of the factors. But if p divides α then $\alpha \equiv 0 \pmod{p}$; that is, $\alpha = 0$ in $\mathbb{Z}_p$. Similarly, if p divides β then $\beta \equiv 0 \pmod{p}$; that is, $\beta = 0$ in $\mathbb{Z}_p$.

To see that the property does not hold in $\mathbb{Z}_n$ when n is not a prime, observe that, if n is not prime, it has the form $n = \alpha\beta$, where $2 \leq \alpha, \beta \leq n - 1$. Hence, α and β are nonzero elements of $\mathbb{Z}_n$, and

$$\alpha\beta = n \equiv 0 (\mathrm{mod} n)$$

that is, $\alpha\beta = 0$ in $\mathbb{Z}_n$. ■

The set $\mathbb{Z}_p$, together with the operations of addition and multiplication modulo p, forms a field. Before giving a formal definition, we need to define the concept of a binary operation on a set.

Definition Let S be a nonempty set. A **binary operation on** S is a function $* : S \times S \rightarrow S$, from the set $S \times S$ of all ordered pairs of elements of S to S. We usually denote $*((\alpha, \beta))$ by $\alpha * \beta$. □

For example, ordinary addition of real numbers is a binary operation $+\mathbb{R} \times \mathbb{R} \to \mathbb{R}$, where $+(\alpha, \beta)$ is almost always written $\alpha + \beta$.

Now let us give a formal definition of a field.

Definition A **field** is a nonempty set $\mathcal{F}$, together with two binary operations on $\mathcal{F}$, called **addition** (denoted by $+$) and **multiplication** (denoted by juxtaposition), satisfying the following properties.

Associative properties: For all α, β, and γ in $\mathcal{F}$,

$$\alpha + (\beta + \gamma) = (\alpha + \beta) + \gamma \text{ and } \alpha(\beta\gamma) = (\alpha\beta)\gamma$$

Commutative properties: For all α and β in $\mathcal{F}$,

$$\alpha + \beta = \beta + \alpha \text{ and } \alpha\beta = \beta\alpha$$

Distributive property: For all α, β and γ in $\mathcal{F}$,

$$\alpha(\beta + \gamma) = \alpha\beta + \alpha\gamma$$

Properties of 0 and 1: There exist two distinct special elements in $\mathcal{F}$, one called the **zero element** and denoted by 0 and the other called the **identity element** and denoted by 1, with the properties that, for all $\alpha \in \mathcal{F}$,

$$0 + \alpha = \alpha + 0 = \alpha \text{ and } 1 \cdot \alpha = \alpha \cdot 1 = \alpha$$

Inverse properties: For every $\alpha \in \mathcal{F}$, there exists another element in $\mathcal{F}$, denoted by $-\alpha$ and called the **negative** of α, for which

$$\alpha + (-\alpha) = (-\alpha) + \alpha = 0$$

For every nonzero $\alpha \in \mathcal{F}$, there exists another element in $\mathcal{F}$, denoted by α^{-1}, and called the **inverse** of α, for which

$$\alpha \cdot (\alpha^{-1}) = (\alpha^{-1}) \cdot \alpha = 1 \qquad \square$$

The most familiar fields are the sets $\mathbb{Q}$ of rational numbers, $\mathbb{R}$ of real numbers, and $\mathbb{C}$ of complex numbers, each with ordinary addition and multiplication. However, it happens that, for every prime power $q = p^n$, there is a field of size q. In fact, there is essentially only one field for each prime power. For $q = p$ a prime, this field is easy to describe, for it is just the integers modulo p.

Theorem 1.1.2 *The set $\mathbb{Z}_n$ of integers modulo n is a field if and only if n is a prime number.*

Proof Suppose first that $n = p$ is a prime. We will not establish all of the properties in the definition of field, but show only that every nonzero element α of $\mathbb{Z}_p$ has an inverse. Consider all possible products of α with the elements of $\mathbb{Z}_p$

$$\{\alpha\beta \mid \beta \in \mathbb{Z}_p\}$$

These p products are distinct, for if $\alpha\beta = \alpha\beta'$ then $\alpha(\beta - \beta') = 0$ and Theorem 0.1.2 gives $\beta - \beta' = 0$, whence $\beta = \beta'$. Since the p products are distinct, they represent every element in $\mathbb{Z}_p$. In particular, one of the products must equal 1, that is, $\alpha\beta = 1$ for some $\beta \in \mathbb{Z}_p$. By commutativity we also have $\beta\alpha = 1$ and so β is the inverse of α.

If n is not a prime, then $n = \alpha\beta$ for some nonzero $\alpha, \beta \in \mathbb{Z}_n$. It follows that $\alpha\beta = 0$ in $\mathbb{Z}_n$ and so α cannot have an inverse. For if α^{-1} did exist, then we would have

$$\beta = \alpha^{-1}(\alpha\beta) = \alpha^{-1}0 = 0$$

which is not the case. Since the nonzero element α does not have an inverse, the set $\mathbb{Z}_n$ cannot be a field. ■

Example 1.1.3 Since $2 + 5 = 0$ in $\mathbb{Z}_7$, the negative of 2 is 5. In symbols, $-2 = 5$. Note that this is true only in $\mathbb{Z}_7$, using addition modulo 7. It is certainly not true in the familiar field of real numbers.

Since $2 \cdot 4 = 1$ in $\mathbb{Z}_7$, the inverse of 2 is 4, that is, $2^{-1} = 4$. Again, this is true only in $\mathbb{Z}_7$, and not in the field of real numbers. □

When p is not a prime, all of the properties in the definition of a field hold except that not all nonzero numbers have inverses. However, since $\mathbb{Z}_n$ is not a field if n is not prime, many other properties that hold in $\mathbb{Z}_p$ do not hold in $\mathbb{Z}_n$. Theorem 1.1.1 is a case in point.

The field $\mathbb{Z}_2$ of integers modulo 2 has an interesting property not shared by the other fields $\mathbb{Z}_p$ for $p > 2$. Since $1 + 1 = 0$ in $\mathbb{Z}_2$, we have $1 = -1$. Certainly $0 = -0$, and so, if e is either 0 or 1, then

$$e - e = e + (-e) = e + e$$

In other words, in $\mathbb{Z}_2$, each element is its own negative and subtraction is the same as addition. Note that this is not true in $\mathbb{Z}_p$, for $p \neq 2$.

We should caution against confusing addition modulo 2 in $\mathbb{Z}_2$ with addition of binary numbers. In $\mathbb{Z}_2$, we have $1 + 1 = 0$, but for addition of binary numbers, we have $1 + 1 = 10$.

Strings

The concept of a string is fundamental to the subjects of coding and information theory. Let $S = \{s_1, s_2, \ldots, s_n\}$ be a finite, nonempty set, which we refer to as an **alphabet**. A **string**, or **word**, over S is simply a finite sequence of elements of S. For instance, if $S = \{\alpha, \beta, 1, 2\}$ then

$$\beta, \alpha 1, 11\beta 2, \text{ and } 2222$$

are strings over S. A sequence does not have to be meaningful in some language to qualify as a word. For instance, if $S = \{a, b, c, \ldots, z\}$, then xyah is a word over S, even though it is not a word in the English language. The terms string and word are synonymous and will be used interchangeably.

Strings will be denoted by boldface letters, such as $\mathbf{x}$, $\mathbf{y}$ and $\mathbf{z}$. If $\mathbf{x} = x_1 x_2 \cdots x_k$ is a string over S, then each x_i in $\mathbf{x}$ is an **element** of $\mathbf{x}$. The **length** of a string $\mathbf{x}$, denoted by $\text{len}(\mathbf{x})$, is the number of elements in the string.

The **juxtaposition** of two strings $\mathbf{x}$ and $\mathbf{y}$ is the string $\mathbf{xy}$. For instance, the juxtaposition of $\mathbf{x} = 101$ and $\mathbf{y} = 1000$ is $\mathbf{xy} = 1011000$. If a string has the form $\mathbf{z} = \mathbf{xy}$, we say that $\mathbf{x}$ is a **prefix** of $\mathbf{z}$. For instance, 110 is a prefix of 1101010. It is clear that

$$\text{len}(\mathbf{xy}) = \text{len}(\mathbf{x}) + \text{len}(\mathbf{y})$$

The set of all strings over S is denoted by S^*. We also include in S^* the **empty string**, denoted by θ (the Greek letter theta), and defined to be the string with no elements. Thus, $\text{len}(\theta) = 0$. If n is a non-negative integer, the set of all strings over S of length n is denoted by S^n, and the set of all strings over S of length n or less is denoted by S_n. Thus, $S^n \subseteq S_n \subseteq S^*$. Note that $S^0 = S_0$ consists of just the empty string.

A string over $\mathbb{Z}_2 = \{0, 1\}$ is called a **binary string**. Each of the elements 0 and 1 is called a **bit**, which is a contraction of binary digit. For instance, 011101 is a binary string of length 6. The **complement** $\mathbf{x}^c$ of a binary string $\mathbf{x}$ is defined to be the string obtained by replacing all 0s by 1s and all 1s by 0s. For instance, $(11001)^c = 00110$. A string over $\mathbb{Z}_3 = \{0, 1, 2\}$ is called a **ternary string**.

If 0 is in the alphabet, then the **stringzero string** $00\cdots0$ is denoted by a boldface $\mathbf{0}$. If 1 is in the alphabet, the string consisting of all 1s is denoted by a boldface $\mathbf{1}$. For instance, in $\mathbb{Z}_2^5$, we have $\mathbf{0} = 00000$ and $\mathbf{1} = 11111$. If $0 \leq i \leq n$, the notation $\mathbf{e}_i$ is reserved strictly for the string all

of whose elements are 0 except for a 1 in the ith position. For instance, in $\mathbb{Z}_2^4$,

$$\mathbf{e}_1 = 1000, \mathbf{e}_2 = 0100, \mathbf{e}_3 = 0010, \mathbf{e}_4 = 0001$$

In $\mathbb{Z}_5$, we have $\mathbf{e}_1 = 10000$. Thus, there is some ambiguity in this notation, since there is a different string $\mathbf{e}_i$ in each $\mathbb{Z}_p^n$ with $n \geq i$. However, this should not cause any problems, since the context will always resolve any ambiguity.

The next theorem tells us how many strings there are in S^n and S_n.

Theorem 1.1.3 *Let S be an alphabet of size $k > 1$. Then*

$$1)\ |S^n| = k^n \qquad 2)\ |S_n| = \frac{k^{n+1} - 1}{k - 1}$$

Proof For part 1), note that each string in S^n can be formed by picking one of the k symbols in S for the first position, one for the second position, one for the third position, and so on. Because there are k choices for each of the n positions, the number of strings that can be formed in this way is k^n. That is, $|S^n| = k^n$.

For part 2), we use the results of part 1),

$$\begin{aligned} |S_n| &= |S^0| + |S^1| + \cdots + |S^n| \\ &= 1 + k + k^2 + \cdots + k^n = \frac{k^{n+1} - 1}{k - 1} \end{aligned}$$

■

Theorem 1.1.4

1. *In $\mathbb{Z}_2^n$, the number of strings with exactly k 0s is $\binom{n}{k}$.*
2. *In $\mathbb{Z}_r^n$, the number of strings with exactly k 0s is $\binom{n}{k}(r-1)^{n-k}$.*

Proof Part 1) follows from the fact that there is one such string for every way of choosing k of the n positions in which to place the 0s. Part 2) is similar, and left as an exercise. ■

Example 1.1.4

1. The set $\mathbb{Z}_2^8$ has size $2^8 = 256$. Furthermore, there are $\binom{8}{3} = 56$ binary strings in $\mathbb{Z}_2^8$ containing exactly three 0s.
2. The set $\mathbb{Z}_3^6$ has size $3^6 = 729$. There are $\binom{6}{2}2^4 = 240$ strings in $\mathbb{Z}_3^6$ that contain exactly two 0s.
3. The number of strings in $\mathbb{Z}_5^{10}$ that contain exactly three 0s and exactly two 1s is $\binom{10}{3}\binom{7}{2}3^5 = 612360$. This follows from the fact that there are

$\binom{10}{3}$ ways to choose the 3 positions for the 0s, then there are $\binom{7}{2}$ ways to choose 2 of the remaining 7 positions for the two 1s and, finally, there are 3^5 ways to fill in the remaining positions with any of the 3 other elements of $\mathbb{Z}_5$.

4. The number of strings in $\mathbb{Z}_7^{12}$ with at least nine 0s is

$$\binom{12}{9}6^3 + \binom{12}{10}6^2 + \binom{12}{11}6 + \binom{12}{12} = 49969 \qquad \square$$

Exercises

1. Write out the addition and multiplication modulo 3 tables for $\mathbb{Z}_3$.
2. Show that the cancellation law

$$\alpha\beta = \alpha\gamma, \alpha \neq 0 \text{ implies } \beta = \gamma$$

holds in $\mathbb{Z}_p$, p a prime. Does it hold in $\mathbb{Z}_n$ when n is not prime?

3. Find the following residues
   a) 23 (mod 11)
   b) -17 (mod 3)
   c) 1345 (mod 5)
   d) 1232456 (mod 2)
   e) 133 (mod 3)
   f) -1793 (mod 5)
4. Find the following inverses
   a) 3^{-1} in $\mathbb{Z}_5$
   b) 3^{-1} in $\mathbb{Z}_7$
   c) 8^{-1} in $\mathbb{Z}_{11}$
5. How many strings are there in $\mathbb{Z}_5^8$ that contain exactly four 0s? How many strings are there that contain exactly three 0s and two 1s?
6. Show that there are $\binom{n}{k}(r-1)^{n-k}$ strings in $\mathbb{Z}_r^n$ containing exactly k 0s
7. How many strings are there in $\mathbb{Z}_3^6$ with at most two 0s?
8. How many strings are there in $\mathbb{Z}_2^{20}$ with at least two 0s?

9. How many strings are there in $\mathbb{Z}_r^n$ that contain exactly k nonzero elements?
10. Prove that $1 + k + k^2 + \cdots + k^n = \frac{1-k^{n+1}}{1-k}$.
11. In $\mathbb{Z}_2^n$, show that
    a) $(\mathbf{x} + \mathbf{y})^c = \mathbf{x} + \mathbf{y}^c$
    b) $\mathbf{x}^c + \mathbf{y}^c = \mathbf{x} + \mathbf{y}$.

For the next exercises, we use the following concept (which will be defined more formally later in the text). The distance between two strings of the same length is the number of positions in which these strings differ. For instance, $d(0011, 0110) = 2$, since these two strings differ in 2 places (the second and fourth).

12. Let $\mathbf{x}$ be a string in $\mathbb{Z}_r^n$. Show that there are $\binom{n}{k}(r-1)^k$ strings in $\mathbb{Z}_r^n$ that have distance k from $\mathbf{x}$.
13. If $\mathbf{x}$ and $\mathbf{y}$ are binary strings of length n, find expressions for
    a) $d(\mathbf{x}^c, \mathbf{y})$
    b) $d(\mathbf{x}^c, \mathbf{y}^c)$
    in terms of $d(\mathbf{x}, \mathbf{y})$.
14. Show that if $\mathbf{x}$, $\mathbf{y}$, and $\mathbf{z}$ are strings of the same length, then

$$d(\mathbf{x}, \mathbf{y}) \leq d(\mathbf{x}, \mathbf{z}) + d(\mathbf{z}, \mathbf{y})$$

This is known as the triangle inequality.

1.2 What Are Codes?

A code is nothing more than a set of strings over a certain alphabet. For example, the set

$$C = \{0, 10, 110, 1110\}$$

is a code over the alphabet $\mathbb{Z}_2$. Of course, codes are generally used to encode messages. For instance, we may use C to encode the first four letters of the alphabet, as follows

$$a \to 0$$
$$b \to 10$$
$$c \to 110$$
$$d \to 1110$$

Then we can encode words (or messages) built up from these letters. The word cab, for instance, is encoded as

$$\text{cab} \to 110010$$

These ideas lead us to make the following definitions.

Definition Let $A = \{a_1, a_2, \ldots, a_r\}$ be a finite set, which we call a **code alphabet**. An **r-ary code** over A is a subset C of the set A^* of all words over A. The elements of C are called **codewords**. The number r is called the **radix** of the code. □

The most commonly used (and studied) alphabet is the set $\mathbb{Z}_2$. Codes over $\mathbb{Z}_2$ are referred to as **binary codes**. Codes over the alphabet $\mathbb{Z}_3$ are referred to as **ternary codes**.

Definition Let $S = \{s_1, s_2, \ldots, s_q\}$ be a finite set, which we refer to as a **source alphabet**. Let C be a code. An **encoding function** is a bijective function $f : S \to C$, from S onto C. If C is a code and $f : S \to C$ is an encoding function, we refer to the ordered pair (C, f) as an **encoding scheme** for S. □

Because an encoding function is bijective (that is, both one-to-one and onto), it associates to each source symbol in the source alphabet one and only one codeword. Moreover, every codeword is associated to a source symbol. This makes it possible to decode any sequence of codewords.

Example 1.2.1 The 26 letters of the alphabet can be encoded as follows. Let the source alphabet be $S = \{a, b, c, \ldots, z\}$, let the code alphabet be $A = \{0, 1, \ldots, 9\}$, and let the code be $C = \{00, 01, 02, \ldots, 25\}$. Let $f : S \to C$ be defined by

$$\begin{aligned} f(a) = 00, f(b) = 01,\ f(c) = 02, f(d) = 03, f(e) = 04,\\ f(f) = 05, f(g) = 06,\ f(h) = 07,\ f(i) = 08,\ f(j) = 09,\\ f(k) = 10,\ f(l) = 11, f(m) = 12, f(n) = 13, f(o) = 14,\\ f(p) = 15, f(q) = 16,\ f(r) = 17,\ f(s) = 18,\ f(t) = 19,\\ f(u) = 20, f(v) = 21, f(w) = 22, f(x) = 23, f(y) = 24,\\ f(z) = 25 \end{aligned}$$

TABLE 1.2.1 The ASCII Encoding Scheme (Partial)

A	→	01000001	J	→	01001010	S	→	01010011
B	→	01000010	K	→	01001011	T	→	01010100
C	→	01000011	L	→	01001100	U	→	01010101
D	→	01000100	M	→	01001101	V	→	01010110
E	→	01000101	N	→	01001110	W	→	01010111
F	→	01000110	O	→	01001111	X	→	01011000
G	→	01000111	P	→	01010000	Y	→	01011001
H	→	01001000	Q	→	01010001	Z	→	01011010
I	→	01001001	R	→	01010010	space	→	00100000

This encoding function may be used to encode any message. For instance,

$$\text{math is fun} \rightarrow 120019070818052013$$

We purposely used two-digit numbers for each codeword. If we had taken the code to be $C' = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, \ldots, 25\}$, then uniqueness problems would have arisen. For instance, the string 1019 could have resulted from several different messages,

$$\text{bat} \rightarrow 1019,\ \text{babj} \rightarrow 1019,\ \text{kt} \rightarrow 1019$$

We will discuss the problem of uniqueness later in this chapter. □

Example 1.2.2 Table 1.2.1 shows a portion of a very commonly used code, known as the **ASCII code**. The acronym ASCII stands for American Standard Code for Information Interchange. This code is used by microcomputers to store characters in memory or on storage media. In this case, the complete source alphabet consists of all upper- and lower-case letters, punctuation marks, and various other symbols. The code consists of all binary numbers from 0000000 to 1111111, that is, from 0 to 127 decimal. (The extended ASCII code, adopted by many personal computers, consists of 8-bit binary numbers.)

For instance, the ASCII code for the upper-case letter A is 1000001 (or 65 decimal). The first thirty-three ASCII codes (not shown in the table) are used for control characters, that is, characters that control the operation of a monitor, printer, or other device. For instance, the decimal number 12 (00001100 binary) is the ASCII code for a form feed, and 7 (00000111 binary) is the ASCII code for tintinnabulation. □

Codes can be divided into two general types as follows.

Definition A **fixed length code**, or **block code**, is a code whose codewords all have the same length n. In this case, the number n is also called the **length** of the code. If a code C contains codewords of varying lengths, it is called a **variable length code**. □

Fixed length codes have advantages and disadvantages over variable length codes. One advantage is that they never require a special symbol to separate the characters in the message being coded. For example, consider the encoded ASCII message

01000011010011110100010001000101

Because the (binary) ASCII code is a fixed length code whose codewords have length 8, we know that the first 8 bits represents the first character of the original message, which according to Table 1.2.1, is C. Similarly, the second set of 8 bits represents the second character in the message, namely O. Continuing in this way, we decode the message to get the word CODE.

Perhaps the main disadvantage of fixed length codes, such as the ASCII code, is that characters that are used frequently, such as the letter e, have codes as long as characters that are used infrequently, such as the space character. On the other hand, variable length codes, which can encode frequently used characters using shorter codewords, can save a great deal of time and space. We will discuss both types of codes in this book.

Exercises

1. Suppose you require a binary block code containing 126 codewords. What is the minimum possible length for this code?
2. Suppose you require a binary block code containing n codewords. What is the minimum possible length for this code?
3. How many encoding functions are possible from the source alphabet $S = \{a, b\}$ to the code $C = \{0, 1\}$? List them.
4. How many encoding functions are possible from the source alphabet $S = \{a, b, c\}$ to the code $C = \{00, 01, 11\}$? List them.

5. Find a formula for the number of encoding functions from a source alphabet of size n to a code of size n.
6. How many r-ary block codes of length n are there over an alphabet A? How many binary codes are there of length 5?
7. How many r-ary codes are there with maximum codeword length n over an alphabet A? What is this number for $r = 2$ (binary codes) and $n = 5$?

1.3 Uniquely Decipherable Codes

One of the most important properties that a code can possess is **unique decipherability**. Informally speaking, this means that any sequence of symbols can be interpreted in at most one way as a sequence of codewords. More formally, we have the following definition.

Definition A code C over an alphabet A is **uniquely decipherable** if, for every string $x_1x_2\cdots x_n$ over A, there exists at most one sequence of codewords $\mathbf{c}_1\mathbf{c}_2\cdots\mathbf{c}_m$ for which

$$\mathbf{c}_1\mathbf{c}_2\cdots\mathbf{c}_m = x_1x_2\cdots x_n$$

Put another way, a code is uniquely decipherable if no two different sequences of codewords represents the same string over A, in symbols, if

$$\mathbf{c}_1\mathbf{c}_2\cdots\mathbf{c}_n = \mathbf{d}_1\mathbf{d}_2\cdots\mathbf{d}_m$$

for codewords $\mathbf{c}_i$ and $\mathbf{d}_j$, then $m = n$ and

$$\mathbf{c}_1 = \mathbf{d}_1, \mathbf{c}_2 = \mathbf{d}_2, \ldots, \mathbf{c}_n = \mathbf{d}_n \qquad \square$$

Example 1.3.1 Consider the following codes

$$C_1 = \{\mathbf{c}_1 = 0, \mathbf{c}_2 = 01, \mathbf{c}_3 = 001\}, \quad C_2 = \{\mathbf{d}_1 = 0, \mathbf{d}_2 = 10, \mathbf{d}_3 = 110\}$$

Code C_1 is not uniquely decipherable, since the string 001 represents either the single codeword $\mathbf{c}_3$ or the string $\mathbf{c}_1\mathbf{c}_2$. On the other hand, C_2 is uniquely decipherable, since any string corresponds to at most one sequence of codewords. For instance, consider the string

1000110

Reading from left to right, we see that 1 is not, by itself, a codeword. But 10 is. Furthermore, only one codeword begins with 10 so 10 must be $\mathbf{d}_2$

$$\begin{array}{l} 10|00110 \\ \mathbf{d}_2| \end{array}$$

Next, we come to a 0, which must represent $\mathbf{d}_1$, since no other codeword begins with 0. Continuing in this way, we see that this string represents only one sequence of codewords.

To prove that C_2 is uniquely decipherable, we will be content with giving a set of observations that show that any given sequence $\mathbf{x}$ of codewords can be interpreted in only one way. In this case, we have the following observations, assuming that $\mathbf{x}$ is read from left to right.

1. If we encounter a 0, this must represent $\mathbf{d}_1$.
2. If we encounter a 1 followed by a 0, this must represent $\mathbf{d}_2$.
3. If we encounter a 1 followed by another 1, the next element must be a 0 and so this must represent $\mathbf{d}_3$. □

Speaking loosely, if a code is uniquely decipherable, then it cannot have very many short codewords. To illustrate this point, if the word 010011 of length 6 is a codeword, then the words 010 and 011 of length 3 cannot both be codewords. We can be more precise about codeword lengths in the following theorem, known as **McMillan's Theorem** (first published in 1956).

Theorem 1.3.1 *(McMillan's Theorem) Let* $C = \{c_1, c_2, \ldots, c_q\}$ *be an r-ary code and let* $\ell_i = \text{len}(c_i)$. *If C is uniquely decipherable, then its codeword lengths* $\ell_1, \ell_2, \ldots, \ell_q$ *must satisfy*

$$\sum_{k=1}^{q} \frac{1}{r^{\ell_k}} \leq 1$$

Proof The following proof is the usual one given for this theorem, although it is not particularly intuitive. Suppose that α_j is the number of codewords in C of length j. Then we have

$$\sum_{k=1}^{q} \frac{1}{r^{\ell_k}} = \sum_{j=1}^{m} \frac{\alpha_j}{r^j},$$

where $m = \max_i\{\ell_i\}$.

Now let u be a positive integer, and consider the quantity

$$\left(\sum_{j=1}^{m} \frac{\alpha_j}{r^j}\right)^u = \left(\frac{\alpha_1}{r} + \frac{\alpha_2}{r^2} + \cdots + \frac{\alpha_m}{r^m}\right)^u$$

Multiplying this out gives

$$= \sum_{\substack{1 \le i_j \le m \\ i_1, i_2, \ldots, i_u}} \frac{\alpha_{i_1}}{r^{i_1}} \cdots \frac{\alpha_{i_u}}{r^{i_u}} = \sum_{\substack{1 \le i_j \le m \\ i_1, i_2, \ldots, i_u}} \frac{\alpha_{i_1}\alpha_{i_2}\cdots\alpha_{i_u}}{r^{i_1+\cdots+i_u}}$$

Now, since $1 \le i_j \le m$, each sum $i_1 + \cdots + i_u$ is at least u and at most um. Collecting terms with a common sum $i_1 + \cdots + i_u$, we get

$$= \sum_{k=u}^{um} \left(\sum_{i_1+\cdots+i_u=k} \alpha_{i_1}\alpha_{i_2}\cdots\alpha_{i_u} \right) \frac{1}{r^k} = \sum_{k=u}^{um} \frac{N_k}{r^k}$$

where

$$N_k = \sum_{i_1+\cdots+i_u=k} \alpha_{i_1}\alpha_{i_2}\cdots\alpha_{i_u}$$

Now we are ready to use the fact that the code is uniquely decipherable. Recalling that α_i is the number of codewords in C of length i, we see that

$$\alpha_{i_1}\alpha_{i_2}\cdots\alpha_{i_u}$$

is the number of possible strings of length $k = i_1 + \cdots + i_u$ consisting of a codeword of length i_1, followed by a codeword of length i_2, and so on, ending with a codeword of length i_u.

Hence, the sum N_k is the total number of strings $\mathbf{c}_1 \cdots \mathbf{c}_u$ of length k made up of exactly u codewords. Since C is uniquely decipherable, no two sequences of u codewords can yield the same string of length k and so there can be at most r^k such sequences of codewords, since r^k is the total number of strings of length k from an r-ary alphabet. In other words,

$$N_k \le r^k$$

and so

$$\left(\sum_{k=1}^{m} \frac{\alpha_k}{r^k}\right)^u \le \sum_{k=u}^{um} \frac{N_k}{r^k} \le \sum_{k=u}^{um} 1 \le um$$

Taking uth roots gives

$$\sum_{k=1}^{m} \frac{\alpha_k}{r^k} \leq u^{1/u} m^{1/u}$$

Since this holds for all positive integers u, we may let u approach ∞. But $u^{1/u} m^{1/u} \to 1$ as $u \to \infty$, and so we must have

$$\sum_{k=1}^{m} \frac{\alpha_k}{r^k} \leq 1 \qquad \blacksquare$$

The inequality in McMillan's Theorem is called **Kraft's Inequality**. McMillan's Theorem confirms that, for a uniquely decipherable code, the codeword lengths must be reasonably large. (The numbers ℓ_i must be large in order to make the terms $1/r^{\ell_i}$ small.)

Example 1.3.2 Suppose we desire a binary code consisting of six codewords, but we restrict the codeword lengths to a maximum of 2. (That is, $\ell_i \leq 2$ for $i = 1, 2, \ldots, 6$.) Since there are precisely six nontrivial strings over $\{0, 1\}$ of length at most 2, our code must consist of these six strings. That is, $C = \{0, 1, 00, 01, 10, 11\}$. But this code is not uniquely decipherable. (The string 01, for example, has two interpretations.)

In this case, the "shortness" of the codewords forces us to use codewords, such as 01, that are made up of smaller codewords (0 and 1). This prevents the code from being uniquely decipherable.

Of course, we could have used McMillan's Theorem to tell us that such a code could not be uniquely decipherable. For, in this case, $\ell_i \leq 2$ for all i. Hence, $r^{\ell_i} \leq r^2$ and so

$$\frac{1}{r^{\ell_i}} \geq \frac{1}{r^2}$$

Thus, since $r = 2$, we have

$$\sum_{k=1}^{6} \frac{1}{r^{\ell_k}} \geq \sum_{k=1}^{6} \frac{1}{r^2} = \sum_{k=1}^{6} \frac{1}{4} = \frac{6}{4} > 1$$

This tells us that Kraft's inequality does not hold for this code, and so it cannot be uniquely decipherable. □

Note that McMillan's Theorem cannot tell us when a particular code is uniquely decipherable, but only when it is not. For the theorem does not say that any code whose codeword lengths satisfy Kraft's inequality must

be uniquely decipherable. Rather, it says that if a code is known to be uniquely decipherable, then its word lengths must satisfy Kraft's inequality. Hence, if a code does not satisfy this inequality, we may conclude that it cannot be uniquely decipherable.

Exercises

1. Is the code $C = \{0, 10, 1100, 1101, 1110, 1111\}$ uniquely decipherable? Justify.
2. Is the code $C = \{0, 10, 110, 1110, 11110, 11111\}$ uniquely decipherable? Justify.
3. Is the code $C = \{0, 01, 011, 0111, 01111, 11111\}$ uniquely decipherable? Justify.
4. Is the code $C = \{0, 10, 110, 1110, 1111, 1101\}$ uniquely decipherable? Justify.
5. Is the code $C = \{0, 10, 1101, 1110, 1011, 110110\}$ uniquely decipherable? Justify.
6. Determine whether or not there is a uniquely decipherable binary code with codeword lengths 1,2,3,3. If so, construct such a code.
7. Determine whether or not there is a uniquely decipherable binary code with codeword lengths 1,3,3,3,4,5,5,5. If so, construct such a code.
8. Is it possible to construct a uniquely decipherable code, over the alphabet $\{0, 1, 2, \ldots, 9\}$, with nine codewords of length 1, nine codewords of length 2, ten codewords of length 3, and ten codewords of length 4?
9. For a given binary code, let $N(k)$ be the total number of sequences of codewords that contain exactly k bits. For instance, if

$$C = \{\mathbf{c}_1 = 0, \mathbf{c}_2 = 10, \mathbf{c}_3 = 11\}$$

then $N(3) = 5$, since the five codeword sequences

$$c_1c_1c_1,\ c_1c_2,\ c_1c_3,\ c_2c_1,\ c_3c_1$$

each contain exactly 3 bits, and no other codeword sequences contain exactly 3 bits.

a) Determine $N(1)$ and $N(2)$.

b) Show that $N(k) = N(k-1) + 2N(k-2)$, for all $k \geq 3$. Hint: a string of length $k \geq 3$ begins with either a codeword of length 1 or a codeword of length 2.

c) Solve the recurrence relation in part b). Hint: assume a solution of the form $N(k) = \alpha^k$ and solve for α. Get a general solution of the form $N(k) = a\alpha_1^k + b\alpha_2^k$ and determine the values of a and b.

d) For the code $D = \{0, 10, 110, 111\}$, compute $N(1)$, $N(2)$, $N(3)$, $N(4)$, $N(5)$. Show that these values are consistent with the formula

$$N(k) = \frac{4}{7} \cdot 2^k + \frac{3}{7} \cos \frac{2\pi}{3}k + \frac{\sqrt{3}}{21} \sin \frac{2\pi}{3}k$$

1.4 Instantaneous Codes and Kraft's Theorem

It is clear that unique decipherability is a very desirable property. However, even though a code may have this property, it may still not be possible to interpret codewords as soon as they are received. The following simple example will illustrate this.

Example 1.4.1 The code

$$C_3 = \{\mathbf{c}_1 = 0, \mathbf{c}_2 = 01\}$$

is easily seen to be uniquely decipherable (by reading strings backwards). Now suppose that the string 0001 is being transmitted. Just after receiving the first 0, we cannot tell whether it should be interpreted as the codeword $\mathbf{c}_1$ or the beginning of $\mathbf{c}_2$. Once we receive the second 0 in the message, we know that the first 0 must represent $\mathbf{c}_1$, but we don't know about the second 0. Thus, codewords cannot be interpreted as soon as they are received.

On the other hand, for the code

$$C_4 = \{\mathbf{d}_1 = 0, \mathbf{d}_2 = 10\}$$

individual codewords can be interpreted as soon as they are received. For instance, consider the string 00100. As soon as the first 0 is received, we know immediately that it must be $\mathbf{d}_1$, and similarly for the second 0.

Example 1.4.4 The code

$$C = \{0, 10, 110, 1110, 11110, 11111\}$$

is an example of a **comma code**. This terminology comes from the fact that the symbol 0 acts as a kind of comma, telling the receiver when a codeword ends. (The receiver can tell when the last codeword ends by its length.)

Comma codes have the prefix property, and so they are instantaneous. On the other hand, consider the code obtained by reversing the order of the bits

$$D = \{0, 01, 011, 0111, 01111, 11111\}$$

This code does not have the prefix property, and so it is not instantaneous. But it is uniquely decipherable, since any sequence of codewords can be deciphered by starting from the end of the message and first picking out all strings of 1s of length 5, which must represent the codeword 11111, then picking out strings of 1s of length 4, and so on. □

Kraft's Theorem

Now we come to a theorem that tells us precisely when an instantaneous code exists with given codeword lengths $\ell_1, \ell_2, \ldots, \ell_q$. This theorem was first published by L. G. Kraft in 1949.

Theorem 1.4.2

1. *(Kraft's Theorem) There exists an instantaneous r-ary code* $C = \{c_1, c_2, \ldots, c_q\}$, *with codeword lengths* $\ell_1, \ell_2, \ldots, \ell_q$, *if and only if these lengths satisfy Kraft's inequality,*

$$\sum_{k=1}^{q} \frac{1}{r^{\ell_k}} \leq 1$$

2. *Let C be an instantaneous r-ary code. Then C is* **maximal instantaneous**, *that is, C is not contained in any strictly larger instantaneous code, if and only if equality holds in Kraft's inequality.*
3. *Suppose that C is an instantaneous code with maximum codeword length m. If C is not maximal, then it is possible to add a word of length m to C without destroying the property of being instantaneous.*

Proof Let C be a code with codeword lengths $\ell_1, \ldots, \ell_q$. We refer to the sum on the left side of Kraft's inequality as Kraft's sum. Let us begin by rewriting Kraft's inequality in a different form. Suppose that C has u_i codewords of length i for $i = 1, \ldots, m$, where m is the maximum codeword length in C. The the Kraft sum can be written

$$\sum_{k=1}^{q} \frac{1}{r^{\ell_k}} = \sum_{i=1}^{m} \frac{u_i}{r^i} = \frac{1}{r^m} \sum_{i=1}^{m} u_i r^{m-i}$$

and Kraft's inequality can be written in the form

$$\sum_{i=1}^{m} u_i r^{m-i} \leq r^m$$

or

$$u_m + \sum_{i=1}^{m-1} u_i r^{m-i} \leq r^m \tag{1.4.1}$$

We can now prove part a). First, we show that an instantaneous code C must satisfy Kraft's inequality. Let $\mathbf{c} \in C$ have length $i \leq m - 1$. Since $\mathbf{c}$ cannot be a prefix of any other codeword in C, none of the words $\mathbf{cx}$, where $\mathbf{x}$ is a string of length $m - i$, can be in C. Since $\text{len}(\mathbf{x}) = m - i$, there are r^{m-i} words of the form $\mathbf{cx}$, all of which must be excluded from C. Moreover, if $\mathbf{d}$ is another codeword, say of length j, then there are an additional r^{m-j} words of length m that must also be excluded from C. (If $\mathbf{cx} = \mathbf{dy}$ for $\mathbf{c} \neq \mathbf{d}$, then one of $\mathbf{c}$ or $\mathbf{d}$ is a prefix of the other, which is not possible.) Thus, the total number of excluded words of length m is precisely equal to the summation on the left side of (1.4.1). Adding the number u_m of words of length m that are in C must result in a number that is no greater than the total number r^m of words of length m. Hence, (1.4.1) holds.

For the converse of part a), we must show that if $\ell_1, \ell_2, \ldots, \ell_q$ satisfy Kraft's inequality, then there is an instantaneous code C with these codeword lengths. This can be proved by induction on the number q. If q is less than or equal to the radix r, then taking distinct code symbols gives an instantaneous code of size q, all of whose codewords have length 1. Certainly, we can extend the length of each codeword to get codewords of lengths $\ell_1, \ldots, \ell_q$, whilst preserving the prefix property. Hence, the result is true for $q \leq r$. Now assume that it is true for all sets of q or

fewer lengths, and let $\ell_1, \ldots, \ell_{q+1}$ be $q+1$ lengths that satisfy Kraft's inequality. We assume, by renumbering if necessary, that $\ell_i \leq \ell_{i+1}$ for all $i = 1, \ldots, q$.

By the induction hypothesis, there is an instantaneous code C of size q with codeword lengths $\ell_1, \ldots, \ell_q$. Moreover, these numbers give strict inequality in Kraft's inequality (1.4.1). Hence, the reasoning that led to (1.4.1) shows that there is at least one word $\mathbf{d}$ of length ℓ_q that has not been included in C, but is also not excluded by virtue of having a codeword in C as prefix. It follows that we may include this word in C and still have an instantaneous code. Lengthening $\mathbf{d}$ (if necessary) by adjoining 0s to the right end will give a codeword of length ℓ_{q+1} without violating the prefix property and so we have an instantaneous code with codeword lengths $\ell_1, \ldots, \ell_{q+1}$.

For part b), suppose first that the codeword lengths of an instantaneous code C give equality in Kraft's inequality. If we add any word to C, the resulting code would not satisfy Kraft's inequality, which implies by what we have just proved above that it cannot be instantaneous. Hence, C is maximal instantaneous.

For the converse, we must show that if a code C is maximal instantaneous, then equality holds in Kraft's inequality. But this follows easily by looking at Kraft's inequality in the form (1.4.1). For if equality does not hold in Kraft's inequality, then the left side of (1.4.1) is less than r^m and so there is at least one word of length m that has not been included in C but is also not excluded by virtue of having a codeword as prefix. Hence, we may include this word in C and still have an instantaneous code. Thus, if C is maximal, equality must hold in Kraft's inequality. This finishes the proof of part b) and also proves part c). ■

Note that Kraft's Theorem says that, if the lengths $\ell_1, \ell_2, \ldots, \ell_q$ satisfy Kraft's inequality, then there must exist some instantaneous code with these codeword lengths. It does not say that any code whose codeword lengths satisfy Kraft's inequality must be instantaneous. As we see in the next example, this is not necessarily the case.

Example 1.4.5 Consider the binary code $C = \{0, 11, 100, 110\}$. The codeword lengths are 1,2, and 3, and since $|A| = 2$, the left side of Kraft's inequality is

$$\frac{1}{2} + \frac{1}{2^2} + \frac{1}{2^3} + \frac{1}{2^3} = 1$$

Hence, these lengths do satisfy Kraft's inequality. Nonetheless, this code is not instantaneous, since the second codeword is a prefix of the fourth. □

Parts b) and c) of Theorem 1.4.2 actually gives us a clue as to how to construct an instantaneous code with given codeword lengths $\ell_1, \ldots, \ell_q$. Suppose that these lengths are arranged so that $\ell_1 \leq \ell_2 \leq \cdots \leq \ell_q$. If we have succeeded in finding $k < q$ codewords $\mathbf{c}_1, \ldots, \mathbf{c}_k$ with lengths $\ell_1, \ldots, \ell_k$, then the Kraft sum, using only these lengths, is strictly less than 1 and so, according to part b) of Theorem 1.4.2, the code $\{\mathbf{c}_1, \ldots, \mathbf{c}_k\}$ is not maximal. Hence, by part c), we may include an additional codeword $\mathbf{c}$ of length ℓ_k or greater, in particular, of length ℓ_{k+1}. The point is that we may add any codeword of length ℓ_{k+1} as long as it does not violate the prefix property, for then we can repeat the process until we have a code of size q. Here is an example.

Example 1.4.6 Let $A = \{0, 1, 2\}$ and let $\ell_1 = 1$, $\ell_2 = 1$, $\ell_3 = 2$, $\ell_4 = 4$, $\ell_5 = 4$, $\ell_6 = 5$. Kraft's inequality is satisfied

$$\frac{1}{3} + \frac{1}{3} + \frac{1}{3^2} + \frac{1}{3^4} + \frac{1}{3^4} + \frac{1}{3^5} = \frac{3^4 + 3^4 + 3^3 + 3 + 3 + 1}{3^5} = \frac{196}{243} < 1$$

and so there exists an instantaneous code C over A with these codeword lengths.

First, we choose the two codewords of the smallest length 1, say

$$\mathbf{c}_1 = 0 \text{ and } \mathbf{c}_2 = 1$$

Then we choose any codeword $\mathbf{c}_3$ of the next smallest length 2 that does not cause the prefix property to be violated. Hence, $\mathbf{c}_2$ cannot start with 0 or 1. Let us choose

$$\mathbf{c}_3 = 20$$

Next we choose any two codewords of length 4 that begin with 2, but not with 20. Let us choose

$$\mathbf{c}_4 = 2100 \text{ and } \mathbf{c}_5 = 2101$$

Finally, we choose any codeword of length 5, not beginning with any previously chosen codeword. We may pick

$$\mathbf{c}_6 = 21100$$

Thus, $C = \{0, 1, 20, 2100, 2101, 21100\}$. Of course, this process is by no means unique and there are other instantaneous codes with these codeword lengths. □

Kraft's Theorem and McMillan's Theorem together tell us something interesting about the relationship between uniquely decipherable codes and instantaneous codes. In particular, if there exists a uniquely decipherable code with codeword lengths $\ell_1, \ell_2, \ldots, \ell_n$, then according to McMillan's Theorem, these lengths must satisfy Kraft's inequality. But then we may apply Kraft's Theorem to conclude that there must also exist an instantaneous code with these lengths. In summary, we have the following remarkable theorem.

Theorem 1.4.3 *If a uniquely decipherable code exits with codeword lengths $\ell_1, \ell_2, \ldots, \ell_n$, then an instantaneous code must also exist with these same codeword lengths.* □

Our interest in this theorem will come later, when we turn to the question of finding desirable codes with the shortest possible codeword lengths. For it tells us that we lose nothing by considering only instantaneous codes (rather than all uniquely decipherable codes).

Let us conclude with another application of Kraft's Theorem.

Example 1.4.7 Let $A = \{0, 1\}$. Suppose that we want an instantaneous code C that contains the codewords 0, 10, and 110. How many additional codewords of length 5 could be added to this code?

Since $|A| = 2$, the three aforementioned codewords contribute

$$\frac{1}{2} + \frac{1}{2^2} + \frac{1}{2^3} = \frac{7}{8}$$

to the sum on the left side of Kraft's inequality. Thus, we have $\frac{1}{8}$ left to work with, so to speak. Now, a codeword of length 5 will contribute $\frac{1}{2^5} = \frac{1}{32}$ to the Kraft sum, and so we cannot add more than four such codewords, since $4 \cdot (\frac{1}{32}) = \frac{1}{8}$. (We may not be able to add as many as four codewords, but we cannot add more than 4.) Checking the possibilities shows that each codeword of length 4 must begin with 111. This leads us to the only possibilities, namely, 11100, 11101, 11110, and 11111. It is not hard to check that we may add these four codewords to our code, that is, that the code

$$\{0, 10, 110, 11100, 11101, 11110, 11111\}$$

is instantaneous. □

Exercises

1. Show that if a code is instantaneous, then it is also uniquely decipherable.
2. Is the code $C = \{0, 10, 1100, 1101, 1110, 1111\}$ instantaneous?
3. Is the code $C = \{0, 10, 110, 1110, 1011, 1101\}$ instantaneous?
4. Can a block code fail to have the prefix property? Explain.
5. Can you construct an instantaneous binary code with codewords 0,10 and an additional nine codewords of length 5? Explain.
6. Find an example of a binary code that is uniquely decipherable but not instantaneous, different from any of the codes in the book.
7. How many prefixes does a word of length n have?

In Exercises 8 through 14, determine whether or not there is an instantaneous code with given radix r and codeword lengths. If so, construct such a code.

8. $r = 2$, lengths 1,2,3,3
9. $r = 2$, lengths 1,2,2,3,3
10. $r = 2$, lengths 1,3,3,3,4,4
11. $r = 2$, lengths 2,2,3,3,4,4,5,5
12. $r = 3$, lengths 1,1,2,2,3,3,3
13. $r = 5$, lengths 1,1,1,1,2,2,2,2,3,3,3,4,4,4
14. $r = 5$, lengths 1,1,1,1,1,8,9
15. Suppose that we want an instantaneous binary code that contains the codewords 0, 10, and 1100. How many additional codewords of length 6 could be added to this code? Construct a code with these additional codewords?
16. Suppose that $\ell_1, \ldots, \ell_q$, and r give equality in Kraft's inequality. Let C be an instantaneous r-ary code with these codeword lengths. If $L = \max\{\ell_i\}$, show that C must contain at least two codewords of maximum length L.

PART I

Information Theory

2 Efficient Encoding

CHAPTER

2.1 Information Sources; Average Codeword Length

In order to achieve unique decipherability, McMillan's Theorem tells us that we must allow reasonably long codewords. Unfortunately, this tends to reduce the efficiency of a code, by requiring longer strings to encode a given amount of data.

On the other hand, it is often the case that not all source symbols occur with the same frequency within a given class of messages. Thus, it makes sense to assign the longer codewords to the less frequently used source symbols, thereby reducing the average number of code symbols per source symbol, and improving the efficiency of the code.

Our plan in this chapter is to construct a certain class of instantaneous encoding schemes that are the most efficient possible among all instantaneous encoding schemes, in a sense that we shall now make precise. (An encoding scheme is instantaneous if the corresponding code is instantaneous.) To this end, we will assume that each source symbol has associated to it a probability of occurrence. This leads us to make the following definition.

Definition An **information source** (or simply **source**) is an ordered pair $\mathcal{S} = (S, \mathcal{P})$, where $S = \{s_1, s_2, \ldots, s_q\}$ is a source alphabet, and $\mathcal{P}$ is a

probability law that assigns to each element s_i of S a probability $\mathcal{P}(s_i)$. The sequence $\mathcal{P}(s_1), \ldots, \mathcal{P}(s_q)$ is the **probability distribution for** $\mathcal{S}$. □

Often we will be interested only in the probability distribution of the source, which we may simply write in the form $P = \{p_1, \ldots, p_q\}$.

A source can be thought of as a "black box" that emits source symbols, one at a time, to form a message. We will assume that the emission of source symbols is independent of time. In other words, the fact that a given source symbol is emitted at a given instant has no effect on which source symbol will be emitted at any other instant.

As a measure of the efficiency of an encoding scheme, we use the average codeword length.

Definition Let $\mathcal{S} = (S, \mathcal{P})$ be an information source, and let (C, f) be an encoding scheme for $S = \{s_1, \ldots, s_q\}$. The **average codeword length** of (C, f) is

$$\sum_{i=1}^{q} \text{len}(f(s_i))\mathcal{P}(s_i) \qquad \square$$

Example 2.1.1 Consider the source alphabet $S = \{\text{a,b,c,d}\}$, with probabilities of occurrence

$$\mathcal{P}(\text{a}) = \frac{2}{17}, \quad \mathcal{P}(\text{b}) = \frac{2}{17}, \quad \mathcal{P}(\text{c}) = \frac{8}{17}, \quad \mathcal{P}(\text{d}) = \frac{5}{17}$$

Consider also the two encoding schemes shown below

Scheme 1	**Scheme 2**
a → 11	a → 01010
b → 0	b → 00
c → 100	c → 10
d → 1010	d → 11

We have

$$\text{Average length for scheme 1} = 2 \cdot \frac{2}{17} + 1 \cdot \frac{2}{17} + 3 \cdot \frac{8}{17} + 4 \cdot \frac{5}{17} = \frac{50}{17}$$

and

$$\text{Average length for scheme 2} = 5 \cdot \frac{2}{17} + 2 \cdot \frac{2}{17} + 2 \cdot \frac{8}{17} + 2 \cdot \frac{5}{17} = \frac{40}{17}$$

Thus, encoding scheme 2 has a smaller average codeword length. In this sense, it is more efficient than scheme 1. □

Example 2.1.2 Table 2.1.1 (see next page) shows the letters of the alphabet and the space character, along with approximate probabilities of occurrence in the English language, based on statistical data. The last three columns of the table show three different encoding schemes.

The first scheme is a simple fixed length code, using the first 27 binary numbers. In this case, the codewords all have length 5, which is the minimum possible codeword length for a fixed length code (since $2^4 < 27 \leq 2^5$). Thus, the average codeword length of this scheme is 5.

The second scheme uses a comma code, discussed in Section 1.4. We have not written out all of the codewords, since their lengths become rather large. (The last two codewords have length 26.) Calculation gives an average codeword length of approximately 7.0607 for this scheme. Hence, the fixed length code is more efficient.

The third scheme is the Huffman encoding scheme. As we will see, Huffman encoding produces the most efficient scheme, in the sense of having the smallest average codeword length, among all instantaneous codes. In this case, a computation shows that the average codeword length is approximately 4.1195, a savings of approximately 18% over the fixed length code.

It is worth noting that the comma code is less efficient than the fixed length code because the probabilities of occurrence are all fairly close to each other. Had the probability of occurrence of the space character, for instance, been much larger compared to the other probabilities, then the comma code would have been more efficient than the fixed length code (but not the Huffman code). □

Exercises

1. Let $S = \{a,b,c,d,e\}$ and $\mathcal{P}(a) = \frac{1}{5}$, $\mathcal{P}(b) = \frac{1}{5}$, $\mathcal{P}(c) = \frac{3}{10}$, $\mathcal{P}(d) = \frac{1}{10}$, $\mathcal{P}(e) = \frac{1}{5}$. Which scheme is more efficient

   (a) $a \to 0$, $b \to 10$, $c \to 110$, $d \to 1110$, $e \to 11110$, or

   (b) $a \to 000$, $b \to 001$, $c \to 010$, $d \to 011$, $e \to 100$?

2. Let $S = \{a,b,c,d,e,f\}$ and $\mathcal{P}(a) = 0.2$, $\mathcal{P}(b) = 0.2$, $\mathcal{P}(c) = 0.3$, $\mathcal{P}(d) = 0.1$, $\mathcal{P}(e) = 0.1$, $\mathcal{P}(f) = 0.1$. Which scheme is more efficient

TABLE 2.1.1

Symbol	Probability	Block Code	Comma code	Huffman code
(Space)	0.1859	00000	0	111
E	0.1031	00001	10	010
T	0.0796	00010	110	1101
A	0.0642	00011	1110	1011
O	0.0632	00100	11110	1001
I	0.0575	00101	111110	0111
N	0.0574	00110	·	0110
S	0.0514	00111	·	0011
R	0.0484	01000	·	0010
H	0.0467	01001		0001
L	0.0321	01010		10101
D	0.0317	01011		10100
U	0.0228	01100		00001
C	0.0218	01101		00000
F	0.0208	01110		110011
M	0.0198	01111		110010
W	0.0175	10000		110001
Y	0.0164	10001		100011
P	0.0152	10010		100010
G	0.0152	10011		100001
B	0.0127	10100		100000
V	0.0083	10101		1100000
K	0.0049	10110		11000011
X	0.0013	10111		1100001011
Q	0.0008	11000		1100001010
J	0.0008	11001	111···10	1100001001
Z	0.0005	11010	111···11	1100001000

(a) a → 0, b → 10, c → 110, d → 1110, e → 11110, f → 111110, or

(b) a → 000, b → 001, c → 010, d → 011, e → 100, f → 101?

3. Assuming a source with a uniform probability distribution, what is the average codeword length of a comma code with ten codewords?

4. Assuming a source with a uniform probability distribution, what is the average codeword length of a comma code with n codewords?

5. How do we minimize the average codeword length of an encoding scheme for a source with a uniform probability distribution?

2.2 Huffman Encoding

In 1952, D. A. Huffman published a method for constructing highly efficient instantaneous encoding schemes. This method is now known as *Huffman encoding*. Before giving an example of Huffman encoding, let us state the reason why this type of encoding is so important. (A proof will be given in the next section.)

Theorem 2.2.1 *Let $\mathcal{S} = (S, \mathcal{P})$ be an information source. Then all Huffman encoding schemes for $\mathcal{S}$ are instantaneous. Furthermore, Huffman encoding schemes have the smallest average codeword length among all instantaneous encoding schemes for $\mathcal{S}$.* □

The minimum average codeword length, taken over all uniquely decipherable r-ary encoding schemes for $\mathcal{S}$ will be denoted by $\text{MinAveCodeLen}_r(\mathcal{S})$. According to Theorem 1.4.3, this is the same as the minimum, taken over all instantaneous encoding schemes. We also denote the average codeword length of any r-ary Huffman encoding scheme for $\mathcal{S}$ by $\text{AveCodeLenHuff}_r(\mathcal{S})$. Then Theorem 2.2.1 can be summarized by writing

$$\text{AveCodeLenHuff}_r(\mathcal{S}) = \text{MinAveCodeLen}_r(\mathcal{S})$$

Now let us give an example of Huffman encoding. Although r-ary Huffman encoding schemes can be constructed for all $r \geq 2$, we will restrict attention to binary Huffman codes. (For information on nonbinary Huffman encoding, we refer the reader to *Coding and Information Theory*, by this author. See also the exercises for this chapter.)

Before reading this example, you should familiarize yourself with the terminology on binary trees in Section 0.1.

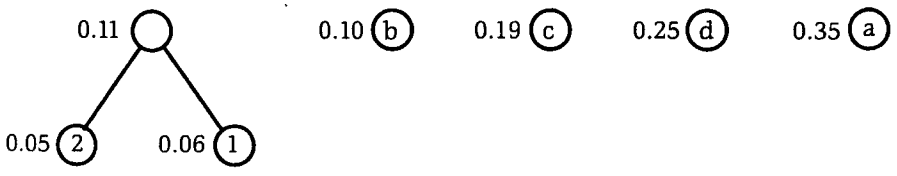
Example 2.2.1 Consider the source alphabet and probabilities shown below.

Symbol	Probability
a	0.35
b	0.10
c	0.19
d	0.25
1	0.06
2	0.05

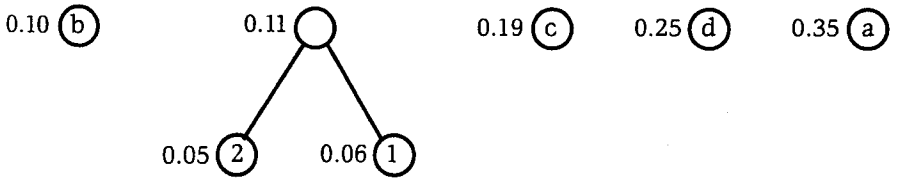
The Huffman encoding scheme is constructed by constructing a complete binary tree as follows.

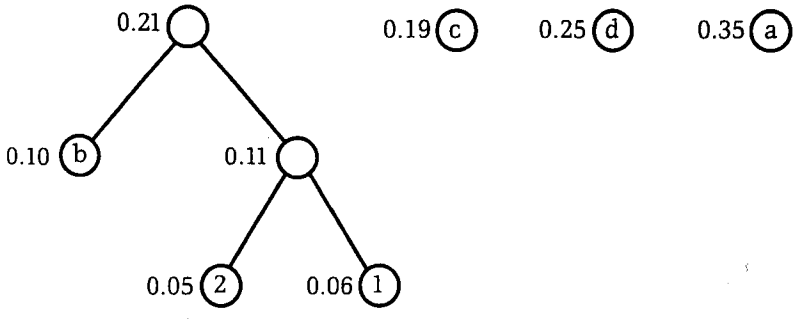
Step 1 Place each symbol inside a node. Then label each node with the probability of occurrence of the symbol and arrange the nodes in order of increasing probability of occurrence.

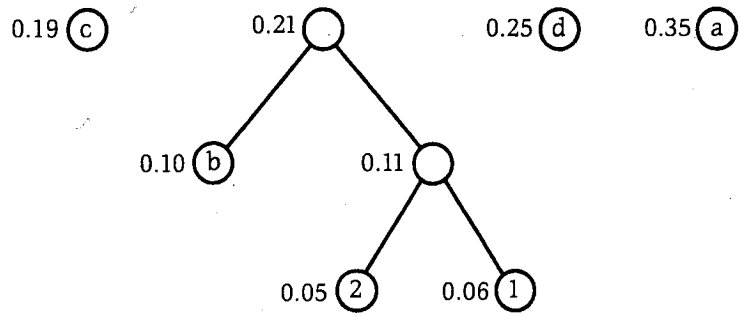
Step 2 Connect the two leftmost nodes to a new node, as shown below. Label the new node with the sum of the probabilities associated to the original nodes. Lower this portion of the figure so that the new node is at the top row.

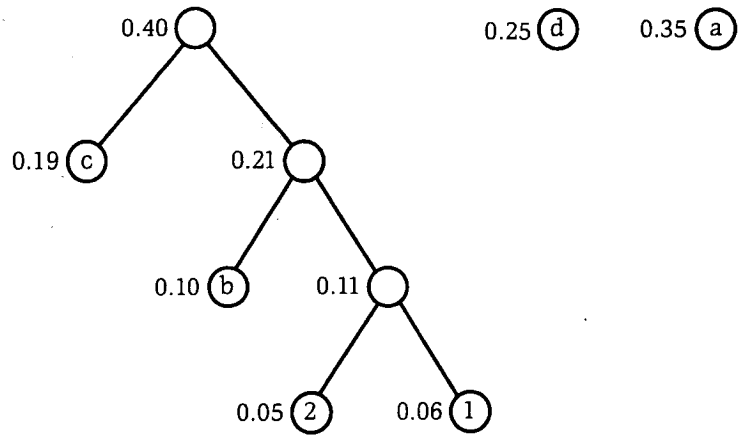


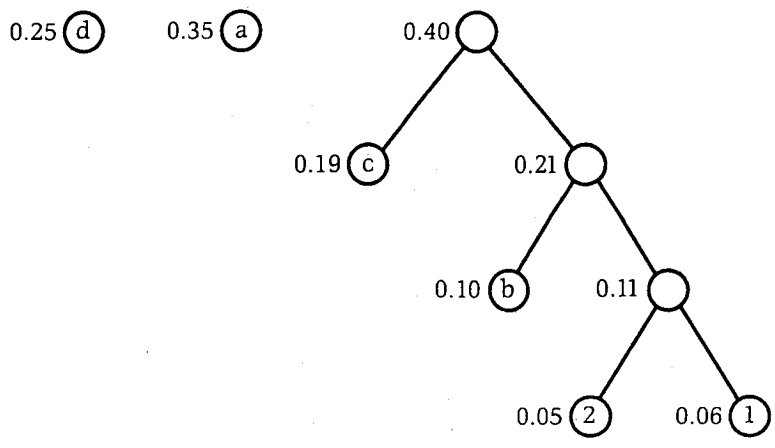
Step 3 Repeat the process of arranging the figure so that the nodes on the top level are in increasing order of probabilities, and then connecting the two leftmost nodes, until only one node remains on the top row. Here are the steps required in this case.

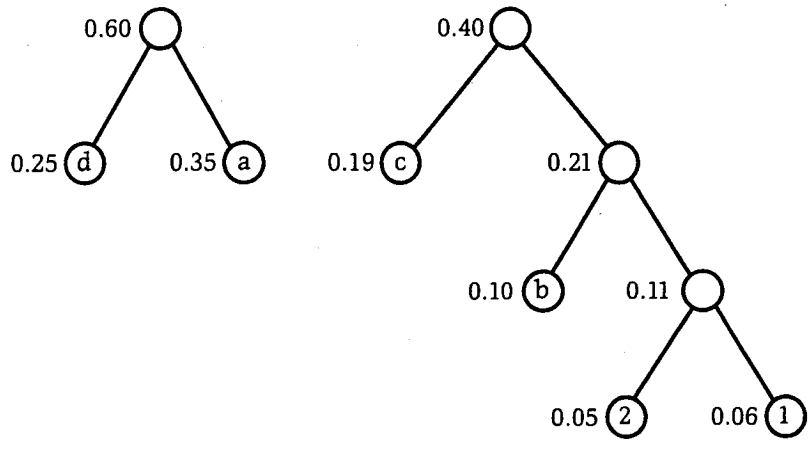





0.19 c
0.21
0.25 d
0.35 a
0.10 b
0.11
0.05 2
0.06 1


0.40
0.25 d
0.35 a
0.19 c
0.21
0.10 b
0.11
0.05 2
0.06 1


0.25 d
0.35 a
0.40
0.19 c
0.21
0.10 b
0.11
0.05 2
0.06 1



Step 4 Discard all of the probabilities, and label each line segment that slants up (from left to right) with a 0 and each line segment that slants down (from left to right) with a 1. This is done in Figure 2.2.2. The result is referred to as a **Huffman tree**.

To determine the codeword associated to each source symbol, start at the root and write down the sequence of bits encountered en route to the

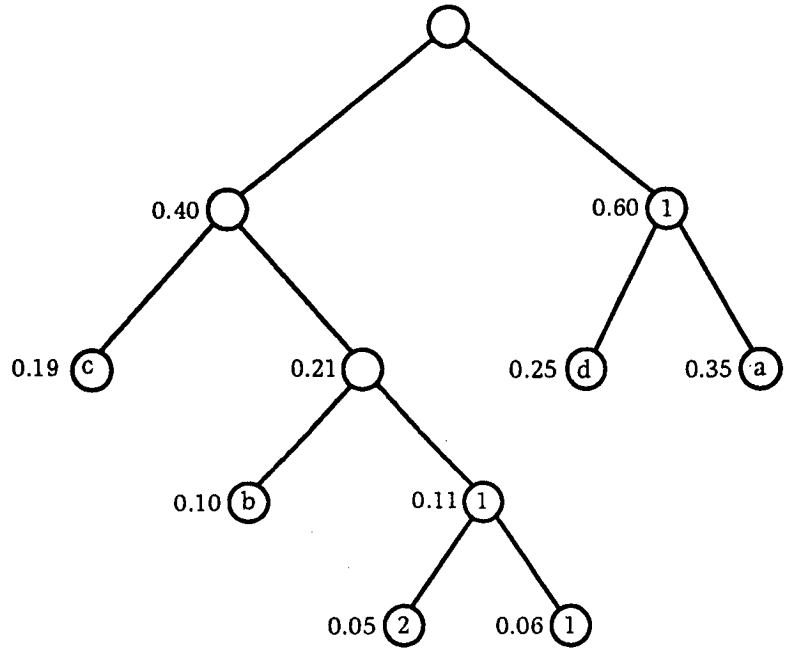


FIGURE 2.2.1

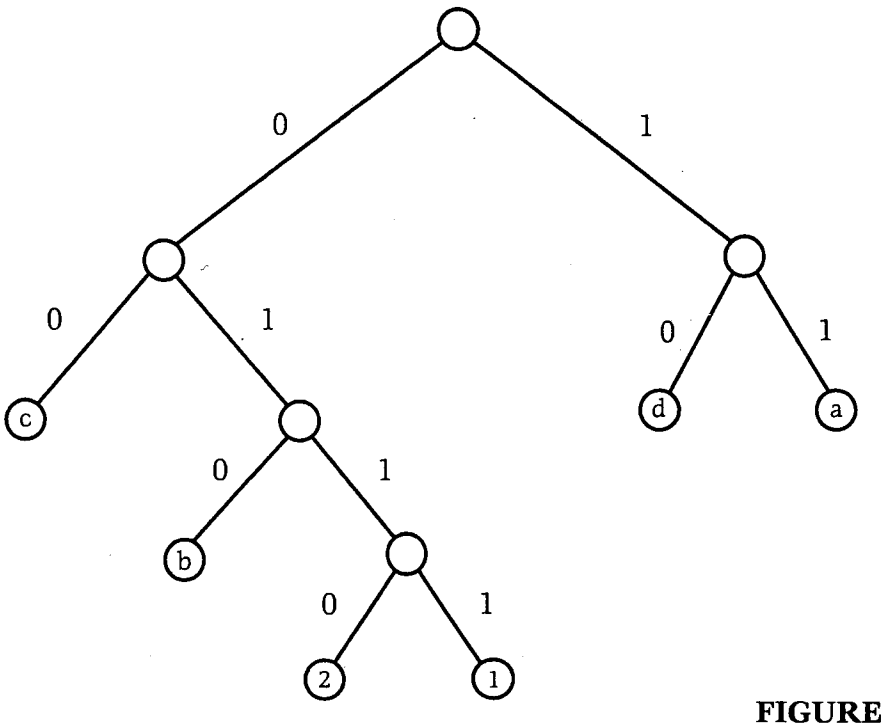


FIGURE 2.2.2

source symbol. In this case, the Huffman encoding is

Source Symbol	Code
a	11
b	010
c	00
d	10
1	0111
2	0110

We will leave it as an exercise to verify that this is an instantaneous code, whose average codeword length is 2.32.

Notice that a binary fixed length code would require codewords of length at least 3 to encode 6 source symbols ($2^2 < 6 \leq 2^3$). Hence, the average codeword length of a fixed length code is 3, and the Huffman code reduces the average codeword length by 22.7%. □

We should remark that the Huffman encoding scheme need not be unique for a given source $\mathcal{S}$. This is due to the ambiguity that occurs when two nodes in the top row have the same probability. Nevertheless, all Huffman encoding schemes for $\mathcal{S}$ have the same average codeword length which, according to Theorem 2.2.1, is the smallest among all instantaneous encoding schemes for $\mathcal{S}$.

Exercises

In Exercises 1–6, find a Huffman encoding of the given probability distribution, using the source symbols $A, B, C, \ldots$ (in this order). Determine the savings over the most efficient fixed length code.

1. $P = \{0.1, 0.2, 0.4, 0.2, 0.1\}$
2. $P = \{0.25, 0.25, 0.25, 0.24, 0.01\}$
3. $P = \{0.1, 0.2, 0.4, 0.1, 0.1, 0.1\}$
4. $P = \{0.05, 0.1, 0.55, 0.05, 0.1, 0.1, 0.05\}$
5. $P = \{0.1, \ldots, 0.1\}$
6. $P = \{0.9, 0.09, 0.009, 0.0009, 0.0001\}$
7. Write a computer program to implement Huffman encoding.
8. State a condition in terms of the sizes of the probabilities that guarantee uniqueness (up to switching 0s and 1s) in Huffman encoding.
9. Determine all source probability distributions $\{p_1, p_2, p_3, p_4\}$ that have $\{00, 01, 10, 11\}$ as Huffman codewords. Hint: think about the Huffman tree.
10. Let (C, f) be a binary Huffman encoding and suppose that the codeword c_i has length ℓ_i for $i = 1, \ldots, k$. Prove that equality holds in Kraft's inequality, that is,

$$\sum \frac{1}{2^{\ell_i}} = 1$$

Hint: Show that there does not exist an instantaneous code D whose codeword lengths m_i satisfy $m_i \leq \ell_i$ for all i and $m_j < \ell_j$ for some j. How does this cause a problem if the Kraft sum is strictly less than 1?

11. (**Huffman codes of radix >2.**) When the radix is greater than 2, Huffman encoding proceeds in a manner entirely analogous to the case $r = 2$, with one exception. In each step, we want to group those r nodes on the top level with smallest probabilities together into a single node, which is labeled with the sum of these r probabilities. However, at the penultimate step, we want exactly r nodes, before combining them into the root node. Thus, the first step may require that we combine fewer than r nodes. Determine the correct number

of nodes to combine into a single node on the first step, so that we may combine r nodes into one on each of the subsequent steps and have exactly r nodes at the penultimate step. How many reduction steps are necessary to complete the Huffman tree?

12. Let (C, f) be a binary Huffman encoding. Let $L = max\{\ell_i\}$. Show that C must contain two codewords $\mathbf{c}$ and $\mathbf{d}$ of maximum length L with the property that they differ only in their last positions.

13. Let (C, f) be a binary Huffman encoding for the uniform probability distribution $P = \{1/n, \ldots, 1/n\}$, and suppose that the codeword lengths of C are ℓ_i.

    (a) Show that (C, f) has minimum total codeword length $T = \sum \ell_i$ among all instantaneous encodings for P.

    (b) Show that C contains two codewords $\mathbf{c}$ and $\mathbf{d}$ of maximum codeword length and that $\mathbf{c}$ and $\mathbf{d}$ differ only in their last positions.

    (c) Show that $\ell_i = L$ or $\ell_i = L - 1$ for all i.

    (d) Let $n = \alpha 2^k$, where $1 < \alpha \leq 2$. Let $\mathbf{u}$ be the number of codewords of length $L - 1$ and let v be the number of codewords of length L. Determine u, v, and L in terms of α and k.

    (e) Find $\text{MinAveCodeLen}_2(\frac{1}{n}, \ldots, \frac{1}{n})$.

14. Given n source symbols and thinking in terms of using frequencies in place of probabilities (which does not affect the results of the Huffman algorithm), what are the minimum possible frequencies (frequencies must be positive integers) to produce a Huffman code with the largest possible maximum codeword length? What is this largest length?

2.3 The Proof that Huffman Encoding Is the Most Efficient

We are now ready to prove the following theorem, first stated in the previous section.

Theorem 2.3.1 *Let S be an information source. Then all Huffman encoding schemes for S are instantaneous. Furthermore, Huffman encoding schemes have the smallest average codeword length among all instantaneous encoding*

schemes for $\mathcal{S}$. In symbols,

$$\text{AveCodeLenHuff}_r(\mathcal{S}) = \text{MinAveCodeLen}_r(\mathcal{S}) \qquad \square$$

Proof Again, we restrict our attention to binary ($r = 2$) codes. The fact that Huffman coding schemes are instantaneous can be seen most easily by considering the Huffman tree. If one codeword was the prefix of another, then it would be possible to get to the second codeword by traveling from the root to the first codeword, and then continuing down the tree from there. But codewords come only at the end nodes (the leaves) of the tree, and so this is not possible. Hence, Huffman encoding schemes have the prefix property, and so they are instantaneous.

Now we must show that Huffman encoding schemes have the smallest average codeword length among all instantaneous encoding schemes. Let (H, f) be a binary Huffman encoding scheme for $\mathcal{S}$, and let (C, g) be any other instantaneous binary encoding scheme. Let us denote the average codeword lengths of these schemes by $\text{AveLen}(H, f)$ and $\text{AveLen}(C, g)$. Thus, we want to prove that

$$\text{AveLen}(H, f) \leq \text{AveLen}(C, g)$$

We begin by making several observations. Table 2.3.1 will set the notation.

Of course, we may assume by reordering the source symbols if necessary that

$$p_1 \geq p_2 \geq \cdots \geq p_q$$

Observation 1 We may assume that the codeword lengths for C satisfy

$$m_1 \leq m_2 \leq \cdots \leq m_q$$

TABLE 2.3.1

Source Symbol	Probability	Huffman Codeword	Huffman Length	Other Codeword	Other Length
s_1	p_1	$\mathbf{h}_1$	ℓ_1	$\mathbf{c}_1$	m_1
s_2	p_2	$\mathbf{h}_2$	ℓ_2	$\mathbf{c}_2$	m_2
s_3	p_3	$\mathbf{h}_3$	ℓ_3	$\mathbf{c}_3$	m_3
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
s_q	p_q	$\mathbf{h}_q$	ℓ_q	$\mathbf{c}_q$	m_q

For if not, then we may interchange two codewords and produce an encoding scheme with smaller average codeword length, which can be used in place of (C, g).

We may also assume that the last two codewords $\mathbf{c}_{q-1}$ and $\mathbf{c}_q$ of C are of equal length, that is, $m_{q-1} = m_q$, and differ only in their last bits. For if not, we may again replace (C, g) by an encoding scheme with smaller average codeword length. To see this, suppose that $m_{q-1} < m_q$, and consider the codeword $\mathbf{c}'_q$, obtained by removing the last bit from $\mathbf{c}_q$. Since C is instantaneous, $\mathbf{c}'_q$ is not a codeword in C and is too long to be a prefix of a codeword in C. Hence, we may replace $\mathbf{c}_q$ by the shorter word $\mathbf{c}'_q$, which will reduce the average codeword length.

Observation 2 During the initial step of Huffman's algorithm, the relative order in which we place source symbols with the same probability of occurrence has no effect on the average codeword length of the encoding scheme, for it amounts to nothing more than a relabeling of source symbols with the same probability of occurrence. Hence, we may arrange it so that the source symbols s_q and s_{q-1} occupy the first two positions on the left (in that order).

It is also clear that, since s_q and s_{q-1} are siblings in the Huffman tree, the codewords $\mathbf{c}_q$ and $\mathbf{c}_{q-1}$ have the same length and, in fact, differ only in their last bits. Hence $\ell_q = \ell_{q-1}$. (See the previous exercise set.)

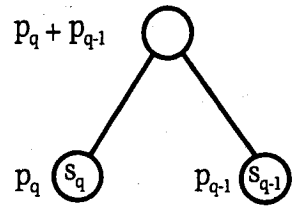
With these observations in mind, a proof can be constructed using induction on the number q of source symbols. If $q = 2$, then the Huffman encoding scheme has average codeword length 1, and so

$$\text{AveLen}(H, f) \leq \text{AveLen}(C, g)$$

Let us assume that the result is true for all source alphabets of size $q - 1$, and then prove that it is also true for source alphabets of size q.

Consider the Huffman scheme (H, f) for the source $\mathcal{S}$. For purposes of induction, we form a new source $\mathcal{S}'$ by replacing s_q and s_{q-1} with a single source symbol s, with probability of occurrence $p_q + p_{q-1}$. This gives us a source alphabet $S' = \{s_1, s_2, \ldots, s_{q-2}, s\}$ of size $q - 1$. To determine the average codeword length for a Huffman encoding (H', f') of this source, observe that, at the second step in the encoding of the original source $\mathcal{S}$,

we connect the two leftmost nodes, to get



From this point on, we will get the same tree by replacing this "minitree" with a single node labeled s, with probability $p_q + p_{q-1}$ and then putting the minitree back at the end. Hence, we can encode the new source $\mathcal{S}'$ by encoding the original source and then replacing the minitree above by the single node s.

Removing this minitree causes the removal of the two nodes for s_q and s_{q-1}, and subtracts

$$\ell_{q-1}p_{q-1} + \ell_q p_q$$

from the average codeword length. But the inclusion of the node for s adds

$$(\ell_{q-1} - 1)(p_{q-1} + p_q)$$

to the average codeword length. Hence, there is a net change of (since $\ell_q = \ell_{q-1}$)

$$(\ell_{q-1} - 1)(p_{q-1} + p_q) - (\ell_{q-1}p_{q-1} + \ell_q p_q) = -(p_{q-1} + p_q)$$

Thus

$$\text{AveLen}(H', f') = \text{AveLen}(H, f) - (p_{q-1} + p_q) \tag{2.3.1}$$

Now consider the encoding scheme (C, g). The last two codewords of C have the form

$$\mathbf{c}_q = x_1 x_2 \cdots x_u 0$$

and

$$\mathbf{c}_{q-1} = x_1 x_2 \cdots x_u 1$$

Since C is instantaneous, the string $\mathbf{c} = x_1 x_2 \cdots x_u$ is not a codeword in C and so we may encode the new source $\mathcal{S}'$ using $\mathbf{c}$ as the codeword for the source symbol s. This results in a net change in average codeword length of (because $m_{q-1} = m_q$)

$$(m_{q-1} - 1)(p_{q-1} + p_q) - (m_{q-1}p_{q-1} + m_q p_q) = -(p_{q-1} + p_q)$$

which is the same as in the Huffman case. Hence,

$$\text{AveLen}(C', g') = \text{AveLen}(C, g) - (p_{q-1} + p_q) \qquad (2.3.2)$$

But, the induction hypothesis implies that

$$\text{AveLen}(H', f') \leq \text{AveLen}(C', g')$$

and this, together with (2.3.1) and (2.3.2), shows that

$$\text{AveLen}(H, f) \leq \text{AveLen}(C, g)$$

as desired. ■

We should conclude by making a few remarks on how Huffman encoding is implemented. Given a message to encode, one does not usually know ahead of time the proper probabilities of occurrence. Using Table 2.1.1 on the relative frequencies of letters in the English language (compiled statistically) may be far from ideal for a given message, and can even lead to a lengthening of some messages.

Accordingly, a static approach is to first scan the message and compile a table of frequencies for each source symbol. Since the Huffman algorithm is not affected by a proportional scaling of the probabilities, these frequences can be used in place of probabilities. (Scaling the frequencies by dividing by their sum would, of course, yield a probability distribution but, in practice, these numbers must be stored in a computer and roundoff errors may actually change the shape of the tree.)

The static approach provides a probability distribution (or set of frequencies) that statistically models the given message, but has the disadvantage that the table of frequencies (or the code itself) must also be transmitted along with the message, for decoding purposes.

A more efficient approach is to use dynamic (also called adaptive) Huffman encoding, which involves scanning the message only once and constantly updating the frequency information after each symbol is encoded. Thus, the probability model (via frequencies) is constantly changing. It is important to emphasize that this approach is entirely outside the theoretical scope of our discussion (both previous and forthcoming), where we assume that a fixed probability distribution is given.

Exercises

1. A complete binary tree is said to be **weighted** if a) each node has a number associated with it, called a **weight**, and b) the weight of a node that is not a leaf is the sum of the weights of its two children. If we divide each weight in a weighted binary tree by the sum of all of the weights, the result is a weighted binary tree the sum of whose weights is 1. Let us refer to such a binary tree as a **normalized weighted complete binary tree**. Prove that a graph G is a Huffman tree (that is, comes from an application of the Huffman algorithm applied to some source) if and only if
   (a) G is a normalized weighted complete binary tree, and
   (b) as we scan the weights of the nodes, going from left to right and starting on the bottom level and proceeding upward through the levels, the weights fall in increasing order by size.

3 Noiseless Coding

CHAPTER

3.1 Entropy

The results of the previous chapter show that Huffman encoding schemes are the most efficient, in the sense of having the smallest average codeword length, among all instantaneous encoding schemes. Our goal in this chapter is to determine just how efficient such an encoding scheme can be. We will see that, to every source $\mathcal{S}$, there is a number, called the *entropy* of $\mathcal{S}$, that has the property that the average codeword length of any instantaneous encoding scheme for $\mathcal{S}$ must be greater than or equal to the entropy of $\mathcal{S}$. In other words, the entropy provides a *lower bound* on the average codeword length of any instantaneous encoding scheme.

The entropy of a source is intended to measure in a precise way the amount of "information" in the source. In order to motivate the concept of the amount of information obtained from a source symbol, let us imagine that a contest is taking place. Each of two contestants has a "black box" that emits source symbols from a source $\mathcal{S}$ with source alphabet $S = \{s_1, s_2\}$, and probabilities $p_1 = \frac{99}{100}$, $p_2 = \frac{1}{100}$. The winner of the contest is the first one to name both source symbols, that is, the first one to have complete information about the set S. (We assume that neither contestant has seen the source symbols beforehand.)

Now, suppose that on the first round, the first contestant gets source symbol s_1, while the second contestant gets s_2. At this point, which contestant is more likely to win the contest?

Since the first contestant still needs to receive source symbol s_2, whose probability of occurrence is $\frac{1}{100}$, whereas the second contestant needs to receive s_1, whose probability of occurrence is $\frac{99}{100}$, it is clear that the second contestant is more likely to win than the first. In some sense then, the second contestant has received more *information* about S from the source symbol s_2, with the smaller probability of occurrence, than did the first contestant. This motivates the statement that, however we decide to define the information obtained from a source symbol, it should have the property that *the less likely a source symbol is to occur, the more information we obtain from an occurrence of that symbol, and conversely.*

Because the information obtained from a source symbol is not a function of the symbol itself, but rather of the symbol's probability of occurrence p, we use the notation $I(p)$ to denote the information obtained from a source symbol with probability of occurrence p. We will make the following reasonable assumptions about the function $I(p)$, defined for all $0 < p \leq 1$.

Assumption 1 $I(p) \geq 0$

Assumption 2 The function $I(p)$ is continuous in p.

Since we assume that the events of s_i and s_j occurring (on different transmissions) are independent, the information obtained from the knowledge that both s_i and s_j have occurred should be the sum of $I(p_i)$ and $I(p_j)$. Since the probability of both events occurring is the product p_ip_j, we get

Assumption 3 $I(p_ip_j) = I(p_i) + I(p_j)$

The remarkable fact about these three assumptions is that there is essentially only one function that satisfies them.

Theorem 3.1.1 *A function $I(p)$, defined for all $0 < p \leq 1$, satisfies the previous three assumptions if and only if it has the form*

$$I(p) = C \lg \frac{1}{p}$$

where C is a positive constant and lg is the logarithm base 2. □

Proof We leave it as an exercise to show that any function of this form satisfies all three assumptions. For the converse, observe first that, by assumption 3,

$$I(p^2) = I(p \cdot p) = I(p) + I(p) = 2I(p)$$

and similarly,

$$I(p^3) = I(p^2 \cdot p) = I(p^2) + I(p) = 3I(p)$$

In general, for any positive integer n,

$$I(p^n) = nI(p) \tag{3.1.1}$$

a statement that can be proved formally by induction. Replacing p by $p^{1/n}$ gives

$$I(p^{1/n}) = \frac{1}{n} I(p) \tag{3.1.2}$$

Since (3.1.1) and (3.1.2) hold for all positive integers n, we have

$$I(p^{n/m}) = \frac{1}{m} I(p^n) = \frac{n}{m} I(p)$$

that is,

$$I(p^q) = qI(p)$$

for all positive rational numbers q.

Since, for any positive real number r, there is a sequence of positive rational numbers q_n for which $\lim_{n\to\infty} q_n = r$, and thus $\lim_{n\to\infty} p^{q_n} = p^r$, the continuity of $I(p)$ implies that

$$I(p^r) = I(\lim_{n\to\infty} p^{q_n}) = \lim_{n\to\infty} I(p^{q_n}) = I(p) \lim_{n\to\infty} q_n = rI(p)$$

Now let us fix a value of p for which $0 < p < 1$. Since any q satisfying $0 < q < 1$ can be written in the form $q = p^{\log_p q}$, we have

$$I(q) = I(p^{\log_p q}) = I(p) \log_p q = C \lg \frac{1}{q}$$

for some constant $C > 0$. Finally, the continuity of the information function gives $I(1) = 0$. ■

Since the arbitrary multiplicative constant can be absorbed in the units of measurement of information, the previous theorem justifies the following definition.

Definition The **information** $I(p)$ obtained from a source symbol s with probability of occurrence $p > 0$, is given by

$$I(p) = \lg \frac{1}{p}$$

where lg is the base 2 logarithm. □

When it is convenient, we will also use the notation $I(s)$ for the information obtained from the source symbol s. However, it is important to keep in mind that the information $I(s)$ depends only on the probability of occurrence of s.

The unit of measurement of information is the bit, which is a contraction of binary unit. The connection between the binary unit and the binary digit (also abbreviated bit) comes from the following observation. If the source is $S = \{0, 1\}$, $P = \{\frac{1}{2}, \frac{1}{2}\}$, then the information given by either source symbol is $I(\frac{1}{2}) = \lg 2 = 1$. In other words, if the source randomly emits 1 binary digit (bit), then the information obtained by a single emission is 1 binary unit (bit).

Example 3.1.1 A personal computer monitor is capable of displaying pictures made up of pixels at a resolution of 1024 columns by 768 rows (and higher). Hence, if each pixel can be in any one of $256 = 2^8$ colors, there are a total of $2^{8\times 1024\times 768} = 2^{6291456}$ different pictures. If each of these pictures is considered to be equally likely, the probability of a given picture occurring is $2^{-6291456}$, and so the information obtained from a single picture is

$$I = \lg 2^{6291456} = 6,291,456 \text{ bits}$$

On the other hand, let us estimate the information obtained from a random speech of 1000 words. (While it is true that most people do not speak in random sequences of words, politicians often do, for example.) A 10,000 word vocabulary would be considered quite excellent (in fact, quite amazing), and the probability of speaking a given sequence of 1000 words from such a vocabulary is 10000^{-1000}. Hence, the amount of information obtained by such a speech is

$$I = \lg 10000^{1000} = 1000 \lg 10000 < 14{,}000 \text{ bits}$$

This proves that a picture is worth more than a thousand words! □

We can now define the concept of entropy.

Definition Let $\mathcal{S} = (S, \mathcal{P})$ be a source, with probability distribution $P = \{p_1, \ldots, p_q\}$. The average information obtained from a single sample from $\mathcal{S}$ is

$$H(\mathcal{S}) = \sum_{i=1}^{q} p_i I(p_i) = \sum_{i=1}^{q} p_i \lg \frac{1}{p_i} = -\sum_{i=1}^{q} p_i \lg p_i$$

The quantity $H(\mathcal{S})$ is called the **entropy** of the source. (When $p_i = 0$, we set $p_i \lg \frac{1}{p_i} = 0$.) Since this quantity depends only on the probability distribution P (and not on the source alphabet S), we also use the notations $H(P)$ and $H(p_1, \ldots, p_n)$ for the entropy. □

Example 3.1.2 Consider a source $\mathcal{S}_1 = (S_1, \mathcal{P}_1)$ for which each source symbol is equally likely to occur, that is, for which $\mathcal{P}_1(s_i) = p_i = 1/q$, for all $i = 1, 2, \ldots, q$. Then

$$H(\mathcal{S}_1) = H(\frac{1}{q}, \ldots, \frac{1}{q}) = \sum_{i=1}^{q} p_i \lg \frac{1}{p_i} = \sum_{i=1}^{q} \frac{1}{q} \lg q = \lg q$$

On the other hand, for a source $\mathcal{S}_2 = (S_2, \mathcal{P}_2)$, where $p_1 = 1$ and $p_i = 0$ for all $i = 2, 3, \ldots, q$, we have

$$H(\mathcal{S}_2) = H(1, 0, \ldots, 0) = p_1 \lg \frac{1}{p_1} = 0$$ □

The previous example illustrates why the entropy of a source is often thought of as a measure of the amount of uncertainty in the source. The source $\mathcal{S}_1$, which emits all symbols with equal probability, is in a much greater state of uncertainty than the source $\mathcal{S}_2$, which always emits the same symbol. Thus, the greater the entropy, the greater the uncertainty in each sample and the more information is obtained from the sample. (The term disorder is also used in this context.)

Example 3.1.3 If $\mathcal{S} = (S, \mathcal{P})$, where $S = \{s_1, s_2, s_3\}$ and

$$\mathcal{P}(s_1) = \frac{1}{4}, \mathcal{P}(s_2) = \frac{1}{4}, \mathcal{P}(s_3) = \frac{1}{2}$$

then the entropy is

$$H(\mathcal{S}) = H(\frac{1}{4}, \frac{1}{4}, \frac{1}{2}) = \frac{1}{4} \lg 4 + \frac{1}{4} \lg 4 + \frac{1}{2} \lg 2 = 1.5$$

This is compared to an entropy of $\lg 3 = 1.585$ for a source of size 3 where each source symbol is equally likely. □

Example 3.1.4 The entropy of the source $\mathcal{S} = (S, \mathcal{P})$ where $S = \{s_1, s_2, s_3\}$ and

$$\mathcal{P}(s_1) = \frac{1}{2}, \mathcal{P}(s_2) = \frac{1}{2}, \mathcal{P}(s_3) = 0$$

is

$$H(\mathcal{S}) = H(\frac{1}{2}, \frac{1}{2}, 0) = \frac{1}{2}\lg 2 + \frac{1}{2}\lg 2 + 0 = \lg 2 = 1$$

This is the same as the entropy of a two-symbol source, each of whose symbols is equally likely. This example illustrates the fact that the addition of a source symbol (or symbols) that cannot occur does not effect the amount of information obtained from a sampling of the source. □

Example 3.1.5 The first two columns of Table 2.1.1 show the information source associated with the letters of the alphabet used in the English language. A computation shows that the entropy for this source is approximately 4.07991. Thus, one gets an average of 4.07991 bits of information by sampling a single letter from English text.

Note that the average codeword length for the Huffman encoding scheme in Table 2.1.1 is approximately 4.1195 bits and so there is a small amount of additional information in the Huffman code beyond what is contained in the source itself. Recall also that no other instantaneous binary code can do better in terms of average codeword length. □

Exercises

1. Compute the entropy of the probability distribution $\{\frac{1}{3}, \frac{2}{3}\}$.
2. Compute the entropy of the probability distribution $\{\frac{1}{8}, \frac{1}{8}, \frac{3}{4}\}$.
3. Compute the entropy of the probability distribution $\{\frac{1}{a}, \frac{1}{a}, \ldots, \frac{2}{a}, \frac{2}{a}\}$ where $a \geq 5$ is an integer.
4. Show that any function of the form $I(p) = C\lg\frac{1}{p}$ satisfies all three assumptions for the entropy function.
5. When is the entropy $H(\mathcal{S})$ of a source equal to 0?
6. Suppose a fair coin is tossed and if the outcome is a heads, we toss it again. How much information do we get if the final outcome is a heads? A tails? How much uncertainty is there in the final outcome?

7. Suppose we toss a fair coin and roll a fair die. Do we get more information (on the average) from this experiment or from the experiment of tossing three fair coins? Four fair coins?

8. How much information do we get (on the average) by sampling from a deck of cards if

   (a) each card is equally likely to be drawn?

   (b) the black cards are twice as likely to be drawn as the red cards?

9. Suppose that we roll a fair die that has two faces numbered 1, two faces numbered 2, and two faces numbered 3. Then we toss a fair coin the number of times indicated by the number on the die and count the number of heads. How much information is obtained (on the average) by this procedure?

3.2 Properties of Entropy

In Example 3.1.2, we saw that the entropy of a source $\mathcal{S}_1$ of size q with uniform probability distribution is equal to $\lg q$, and that the entropy of a source $\mathcal{S}_2$ where one symbol has probability of occurrence 1 is equal to 0. These are the two "extreme" cases for the value of the entropy of any source. In other words, the entropy satisfies $0 \leq H(\mathcal{S}) \leq \lg q$ for all sources of size q.

In order to prove this fact, we must first establish some preliminary results concerning logarithms, whose proofs are left as exercises.

Lemma 3.2.1

1. *If* ln *denotes the natural logarithm then, for all* $x > 0$,

$$\ln x \leq x - 1$$

2. *If* lg *denotes the logarithm base 2 then, for all* $x > 0$,

$$\lg x \leq \frac{x-1}{\ln 2}$$

In both cases, equality holds if and only if $x = 1$. □

Lemma 3.2.2 *Let $P = \{p_1, p_2, \ldots, p_q\}$ be a probability distribution. Let $R = \{r_1, r_2, \ldots, r_q\}$ have the property that $0 \le r_i \le 1$ for all i, and*

$$\sum_{i=1}^{q} r_i \le 1$$

(Note the inequality here.) Then

$$\sum_{i=1}^{q} p_i \lg \frac{1}{p_i} \le \sum_{i=1}^{q} p_i \lg \frac{1}{r_i}$$

with equality holding if and only if $p_i = r_i$ for all i. □

Proof According to Lemma 3.2.1,

$$\begin{aligned}
\sum_{i=1}^{q} p_i \lg \frac{r_i}{p_i} &\le \frac{1}{\ln 2} \sum_{i=1}^{q} p_i \left(\frac{r_i}{p_i} - 1 \right) \\
&= \frac{1}{\ln 2} \sum_{i=1}^{q} (r_i - p_i) \\
&= \frac{1}{\ln 2} \left(\sum_{i=1}^{q} r_i - \sum_{i=1}^{q} p_i \right) \\
&= \frac{1}{\ln 2} \left(\sum_{i=1}^{q} r_i - 1 \right) \le 0
\end{aligned}$$

Thus

$$\sum_{i=1}^{q} p_i \lg \frac{r_i}{q_i} \le 0$$

Writing $\lg(r_i/p_i) = \lg(1/p_i) - \lg(1/r_i)$ and rearranging gives

$$\sum_{i=1}^{q} p_i \lg \frac{1}{p_i} \le \sum_{i=1}^{q} p_i \lg \frac{1}{r_i}$$

Finally, equality holds here if and only if it holds in Lemma 3.2.1, which happens if and only if $r_i/p_i = 1$ for all i. ■

With these lemmas at our disposal, we can prove the main result of this section.

Theorem 3.2.3 *For a source $\mathcal{S} = (S, \mathcal{P})$ of size q, the entropy $H(\mathcal{S})$ satisfies*

$$0 \le H(\mathcal{S}) \le \lg q$$

Furthermore, $H(\mathcal{S}) = \lg q$ if and only if all of the source symbols are equally likely to occur, and $H(\mathcal{S}) = 0$ if and only if one of the source symbols has probability 1 of occurring. □

Proof Let $P = \{p_1, \ldots, p_q\}$ be the probability distribution of $\mathcal{S}$ and let $R = \{1/q, 1/q, \ldots, 1/q\}$ be the uniform distribution. Applying Lemma 3.2.2 to P and R gives

$$\begin{aligned} H(\mathcal{S}) &= \sum_{i=1}^{q} p_i \lg \frac{1}{p_i} \leq \sum_{i=1}^{q} p_i \lg \frac{1}{1/q} \\ &= \sum_{i=1}^{q} p_i \lg q = (\lg q) \sum_{i=1}^{q} p_i = \lg q \end{aligned}$$

Thus, $H(\mathcal{S}) \leq \lg q$. As for equality, this happens precisely when equality holds in Lemma 3.2.2, that is, when $p_i = 1/q$ for all i. Proof of the final statement is left as an exercise. ■

Theorem 3.2.3 confirms the fact that, on the average, the most information is obtained from sources for which each source symbol is equally likely to occur.

Let us examine a bit more closely the entropy of the special **binary source** $S = \{0, 1\}$, with probability distribution of the form $P = \{p, 1-p\}$. Thus, the entropy of a binary source is

$$H(\mathcal{S}) = p \lg \frac{1}{p} + (1-p) \lg \frac{1}{1-p}$$

The function on the right is often denoted by $H(p)$

$$H(p) = p \lg \frac{1}{p} + (1-p) \lg \frac{1}{1-p} \tag{3.2.1}$$

and called the *entropy function*. Its graph is shown in Figure 3.2.1. (Note that $p \lg(\frac{1}{p})$ is defined to be 0 when $p = 0$.) As expected, the entropy function reaches its maximum value when $p = 1 - p = 1/2$.

A final note. The definition of entropy involves base 2 logarithms, but it is sometimes convenient to use logarithms to other bases. Accordingly, for any positive integer r, we define the **r-ary entropy of a source** $\mathcal{S}$ by

$$H_r(\mathcal{S}) = \sum_{i=1}^{q} p_i \log_r \frac{1}{p_i}$$

Thus, the entropy $H(\mathcal{S}) = H_2(\mathcal{S})$ is the binary entropy.

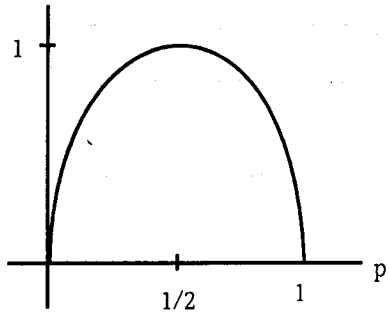


FIGURE 3.2.1 The entropy function $H(p)$

Exercises

1. Prove Lemma 3.2.1.
2. Compute the derivative of the entropy function $H(p)$ given in (3.2.1).
3. Prove that the entropy function $H(p)$ in (3.2.1) is symmetric about the line $x = \frac{1}{2}$.
4. Show that

$$H_r(\mathcal{S}) = \frac{H(\mathcal{S})}{\lg r}$$

5. Find a relationship between $H_r(\mathcal{S})$ and $H_s(\mathcal{S})$.
6. Let $P = \{p_1, \ldots, p_n\}$ be a probability distribution. Suppose that ϵ is a positive real number and that $p_1 - \epsilon > p_2 + \epsilon \geq 0$. Thus, $\{p_1 - \epsilon, p_2 + \epsilon, \ldots, p_n\}$ is also a probability distribution. Interpret the inequality

$$H(p_1, \ldots, p_n) < H(p_1 - \epsilon, p_2 + \epsilon, p_3, \ldots, p_n)$$

   in words. Verify this inequality.

7. Use Lemma 3.2.2 to prove that, if $\{p_1, \ldots, p_n\}$ is a probability distribution, then

$$x_1^{p_1} \cdots x_n^{p_n} \leq p_1 x_1 + \cdots + p_n x_n$$

   where $x_1, \ldots, x_n$ are positive real numbers. This says that the geometric mean of the x_i is less than or equal to the arithmetic mean. Prove that equality holds if and only if the x_i are all equal. Hint: consider the expressions $r_i = p_i x_i / \sum_j p_j x_j$.

3.3 Extensions of an Information Source

Consider the binary source $S = \{s_1, s_2\}$, with probabilities

$$\mathcal{P}(s_1) = p_1 = 0.25, \mathcal{P}(s_2) = p_2 = 0.75$$

A Huffman encoding for this source is

$$s_1 \rightarrow 0$$
$$s_2 \rightarrow 1$$

with average codeword length 1.

Rather than encoding each symbol from S, suppose we encode all strings of length two over S. In other words, consider the source with alphabet

$$S^2 = \{s_1s_1, s_1s_2, s_2s_1, s_2s_2\}$$

where the probabilities of occurrence are determined by multiplication,

$$\mathcal{P}(s_1s_1) = p_1p_1 = (0.25)(0.25) = 0.0625$$
$$\mathcal{P}(s_1s_2) = p_1p_2 = (0.25)(0.75) = 0.1875$$
$$\mathcal{P}(s_2s_1) = p_2p_1 = (0.75)(0.25) = 0.1875$$
$$\mathcal{P}(s_2s_2) = p_2p_2 = (0.75)(0.75) = 0.5625$$

The Huffman algorithm gives the encoding

$$s_1s_1 \rightarrow 010$$
$$s_1s_2 \rightarrow 011$$
$$s_2s_1 \rightarrow 00$$
$$s_2s_2 \rightarrow 1$$

This scheme has average codeword length

$$(0.0625) \cdot 3 + (0.1875) \cdot 3 + (0.1875) \cdot 2 + (0.5625) \cdot 1 = 1.6875$$

But since each codeword represents two source symbols, the average codeword length per original source symbol is $1.6875/2 = 0.84375$, which is an improvement over encoding the original source. Continuing this theme, let S^3 be the source alphabet consisting of strings of length 3 over S. Each source symbol in S^3 is assigned a probability as before. For

instance,

$$\mathcal{P}(s_1s_2s_1) = p_1p_2p_1 = (0.25)(0.75)(0.25) = 0.046875$$

A Huffman encoding of this source is

$$\begin{aligned}
s_1s_1s_1 &\to 11100\\
s_1s_1s_2 &\to 11101\\
s_1s_2s_1 &\to 11110\\
s_1s_2s_2 &\to 100\\
s_2s_1s_1 &\to 11111\\
s_2s_1s_2 &\to 101\\
s_2s_2s_1 &\to 110\\
s_2s_2s_2 &\to 0
\end{aligned}$$

which has an average codeword length of 2.46875, or an average codeword length per original source symbol of $2.46875/3 = 0.82292$, which is an additional improvement over the original encoding.

From these examples, we see that it may be possible to improve the average codeword length per original source symbol by grouping source symbols to form a new source. While it is true that, in some cases, this method does not result in improvements, the method is important, and does lead, as we shall see, to significant theoretical results. This leads us to make the following definition.

Definition Let $\mathcal{S} = (S, \mathcal{P})$ be an information source. The **nth extension** of $\mathcal{S}$ is the source $\mathcal{S}^n = (S^n, \mathcal{P}^n)$, where S^n is the set of all words of length n over S, and $\mathcal{P}^n$ is the probability distribution defined as follows. If $s = s_{i_1}s_{i_2}\cdots s_{i_n}$ is a word in S^n, then

$$\mathcal{P}^n(s) = \mathcal{P}(s_{i_1}s_{i_2}\cdots s_{i_n}) = p_{i_1}p_{i_2}\cdots p_{i_n} \qquad \square$$

The entropy of an extension $\mathcal{S}^n$ is related to the entropy of $\mathcal{S}$ in a very simple way. In fact, when we think of entropy as the average amount of information obtained per symbol, it seems intuitively clear that, since we get n times as much information from a word of length n as from a single character, the entropy of $\mathcal{S}^n$ should be n times the entropy of $\mathcal{S}$. The following theorem confirms this.

Theorem 3.3.1 *Let $\mathcal{S}$ be an information source, and let $\mathcal{S}^n$ be its nth extension. Then $H(\mathcal{S}^n) = nH(\mathcal{S})$.* $\square$

Proof The entropy of the nth extension is

$$H(\mathcal{S}^n) = \sum_{\substack{i_1,i_2,\ldots,i_n \\ 0\le i_k\le q}} p_{i_1}p_{i_2}\cdots p_{i_n} \lg \frac{1}{p_{i_1}p_{i_2}\cdots p_{i_n}}$$

The properties of logarithms give

$$\begin{aligned} H(\mathcal{S}^n) = &\sum_{i_1,i_2,\ldots,i_n} p_{i_1}p_{i_2}\cdots p_{i_n} \lg \frac{1}{p_{i_1}} \\ &+ \sum_{i_1,i_2,\ldots,i_n} p_{i_1}p_{i_2}\cdots p_{i_n} \lg \frac{1}{p_{i_2}} \\ &+ \cdots + \sum_{i_1,i_2,\ldots,i_n} p_{i_1}p_{i_2}\cdots p_{i_n} \lg \frac{1}{p_{i_n}} \end{aligned} \tag{3.3.1}$$

Now, let us look at the first of these summations

$$\sum_{i_1,i_2,\ldots,i_n} p_{i_1}p_{i_2}\cdots p_{i_n} \lg \frac{1}{p_{i_1}} = \sum_{i_1=1}^{q} p_{i_1} \lg \frac{1}{p_{i_1}} \times \sum_{i_2=1}^{q} p_{i_2} \times \cdots \times \sum_{i_n=1}^{q} p_{i_n}$$

Since the sum of the probabilities p_j equals 1, this equals

$$\sum_{i_1=1}^{q} p_{i_1} \lg \frac{1}{p_{i_1}} = H(\mathcal{S})$$

Since each of the other sums in the expression (3.3.4) for $H(\mathcal{S}^n)$ is also equal to $H(\mathcal{S})$, and since there are n such sums, we get

$$H(\mathcal{S}^n) = H(\mathcal{S}) + H(\mathcal{S}) + \cdots + H(\mathcal{S}) = nH(\mathcal{S}) \qquad \blacksquare$$

Example 3.3.1 The entropy of the binary source $S = \{s_1, s_2\}$, $p_1 = 0.25$, $p_2 = 0.75$, is

$$H(\mathcal{S}) = 0.25 \lg \frac{1}{0.25} + 0.75 \lg \frac{1}{0.75} = 0.81128$$

Hence, the entropy of the nth extension $\mathcal{S}^n$ is

$$H(\mathcal{S}^n) = nH(\mathcal{S}) = 0.81128n$$

As we will see in the next section, there is a simple relationship between the entropy of a source (or its extensions) and the average codeword length of any Huffman encoding of that source (or its extensions.) □

Exercises

1. Consider the $\mathcal{S}$ source with alphabet $S = \{a, b\}$ and probability distribution $\mathcal{P}(a) = \frac{1}{4}, \mathcal{P}(b) = \frac{3}{4}$. Construct a Huffman encoding scheme for $\mathcal{S}$, $\mathcal{S}^2$, and $\mathcal{S}^3$ and find the average codeword lengths per source symbol.
2. Repeat the previous exercise with a uniform probability distribution on S.

3.4 The Noiseless Coding Theorem

When encoding a source $\mathcal{S}$, it certainly seems reasonable that we will need at least as many bits of information in the encoding as there is in the source. (For efficient encoding, we also want as few extra bits in the encoding as possible.) Since the entropy of $\mathcal{S}$ measures the amount of information in $\mathcal{S}$, it should come as no surprise that the minimum average codeword length of any encoding of $\mathcal{S}$ should be at least as great as the entropy of $\mathcal{S}$. In symbols,

$$H_r(\mathcal{S}) \leq \text{MinAveCodeLen}_r(\mathcal{S})$$

This is the content of part of the famous Noiseless Coding Theorem, first proved by Claude Shannon in 1948. (Noise refers to the introduction of errors in the code.)

Theorem 3.4.1 (The Noiseless Coding Theorem—Version 1) *Let $\mathcal{S}$ be an information source. Then*

$$H_r(\mathcal{S}) \leq \text{MinAveCodeLen}_r(\mathcal{S})$$

where $\text{MinAveCodeLen}_r(\mathcal{S})$ *denotes the minimum average codeword length among all uniquely decipherable r-ary encoding schemes for $\mathcal{S}$.* □

Proof Denote the probability distribution of the source $\mathcal{S}$ by $P = \{p_1, p_2, \ldots, p_q\}$. Let (C, f) be a uniquely decipherable r-ary encoding scheme for $\mathcal{S}$, with codeword lengths $\ell_1, \ell_2, \ldots, \ell_q$ and consider the numbers

$$r_i = \frac{1}{r^{\ell_i}}$$

The r_i satisfy $0 \leq r_i \leq 1$. Furthermore, since C is uniquely decipherable, McMillan's Theorem tells us that

$$\sum_{i=1}^{q} r_i = \sum_{i=1}^{q} \frac{1}{r^{\ell_i}} \leq 1$$

Thus, Lemma 3.2.2 implies that

$$H(\mathcal{S}) = \sum_{i=1}^{q} p_i \lg \frac{1}{p_i} \leq \sum_{i=1}^{q} p_i \lg \frac{1}{r_i} = \sum_{i=1}^{q} p_i \lg r^{\ell_i} = \sum_{i=1}^{q} p_i \ell_i \lg r$$

$$= \lg r \sum_{i=1}^{q} p_i \ell_i = (\lg r)\text{AveCodeLen}(C, f)$$

Dividing by $\lg r$, and noting that $H_r(\mathcal{S}) = H(\mathcal{S})/\lg r$, we get

$$H_r(\mathcal{S}) \leq \text{AveCodeLen}(C, f)$$

Since this holds for any uniquely decipherable r-ary encoding scheme for $\mathcal{S}$, the result follows. ■

Example 3.4.1 Consider the source $\mathcal{S} = (S, \mathcal{P})$, where $S = \{0, 1, \ldots, 9\}$ and $\mathcal{P}$ is uniform. The entropy of this source is $\lg 10$. According to the Noiseless Coding Theorem, the average codeword length of any uniquely decipherable *ternary* encoding scheme (alphabet of size 3) must be at least

$$H_3(\mathcal{S}) = \frac{H(\mathcal{S})}{\lg 3} = 2.0959 \qquad (3.4.1) \quad \square$$

Example 3.4.2 Table 2.1.1 contains an information source corresponding to the letters of the English language. In Example 3.1.5, we noted that the entropy of this source is approximately 4.07991, and so the Noiseless Coding Theorem tell us that any uniquely decipherable encoding scheme must have average codeword length of at least 4.07991.

Table 2.1.1 also shows a Huffman encoding scheme for this source. In Example 2.1.2, we mentioned that the average codeword length of this Huffman encoding scheme is approximately 4.1195, which is quite close to the minimum possible. □

The first version of the Noiseless Coding Theorem says that the entropy $H_r(\mathcal{S})$ provides a *lower bound* on $\text{MinAveCodeLen}_r(\mathcal{S})$. Let us now turn to the issue of finding an *upper bound* on $\text{MinAveCodeLen}_r(\mathcal{S})$. For this, we wish to construct an instantaneous encoding of $\mathcal{S}$ with

small codeword lengths. Recall that if the lengths $\ell_1, \ldots, \ell_q$ satisfy Kraft's inequality

$$\sum_{i=1}^{q} \frac{1}{r^{\ell_i}} \leq 1$$

then there is an instantaneous code with these codeword lengths. If $P = \{p_1, \ldots, p_q\}$ is the probability distribution for $\mathcal{S}$, then Kraft's inequality can be written in the form

$$\sum_{i=1}^{q} \frac{1}{r^{\ell_i}} \leq \sum_{i=1}^{q} p_i$$

Thus, if

$$\frac{1}{r^{\ell_i}} \leq p_i$$

for all i, Kraft's inequality will be satisfied. This can be rewritten in the form

$$\log_r \frac{1}{p_i} \leq \ell_i$$

so let us choose ℓ_i to be the *smallest* integer satisfying this inequality. In other words, if the integers ℓ_i are chosen to satisfy

$$\log_r \frac{1}{p_i} \leq \ell_i < \log_r \frac{1}{p_i} + 1 \tag{3.4.1}$$

for all i, then there is an instantaneous encoding with these codeword lengths. An encoding scheme whose codeword lengths ℓ_i satisfy (3.4.2) is referred to as a **Shannon-Fano encoding scheme**. Moreover,

$$\begin{aligned} \text{AveCodeLen}_r(\mathcal{S}) &= \sum_{i=1}^{q} p_i \ell_i < \sum_{i=1}^{q} p_i \left(\log_r \frac{1}{p_i} + 1 \right) \\ &= \sum_{i=1}^{q} p_i \log_r \frac{1}{p_i} + \sum_{i=1}^{q} p_i = H_r(\mathcal{S}) + 1 \end{aligned}$$

Hence,

$$\text{AveCodeLen}_r(\mathcal{S}) < H_r(\mathcal{S}) + 1$$

from which it follows that

$$\text{MinAveCodeLen}_r(\mathcal{S}) < H_r(\mathcal{S}) + 1$$

Combining this upper bound with the first version of the Noiseless Coding Theorem gives the second version of this theorem.

Theorem 3.4.2 (The Noiseless Coding Theorem—Version 2) *Let $\mathcal{S}$ be an information source. Then*

$$H_r(\mathcal{S}) \leq \text{MinAveCodeLen}_r(\mathcal{S}) < H_r(\mathcal{S}) + 1$$

where $\text{MinAveCodeLen}_r(\mathcal{S})$ is the minimum average codeword length among all uniquely decipherable r-ary encoding schemes for $\mathcal{S}$. □

This theorem tells us that $\text{MinAveCodeLen}_r(\mathcal{S})$ lies between $H_r(\mathcal{S})$ and $H_r(\mathcal{S}) + 1$. However, the difference between these bounds is 1 r-ary unit per source symbol, and this is still quite a lot from a practical standpoint. Fortunately, better results can be achieved by considering the encoding of extensions of the source $\mathcal{S}$.

In particular, since the nth extension $\mathcal{S}^n$ of $\mathcal{S}$ is a source in its own right, we may apply the Noiseless Coding Theorem to $\mathcal{S}^n$, to get

$$H_r(\mathcal{S}^n) \leq \text{MinAveCodeLen}_r(\mathcal{S}^n) < H_r(\mathcal{S}^n) + 1$$

But $H_r(\mathcal{S}^n) = nH_r(\mathcal{S})$, and so

$$nH_r(\mathcal{S}) \leq \text{MinAveCodeLen}_r(\mathcal{S}^n) < nH_r(\mathcal{S}) + 1$$

Dividing by n gives the final version of the Noiseless Coding Theorem.

Theorem 3.4.3 (The Noiseless Coding Theorem—Final Version) *Let $\mathcal{S}$ be an information source, and let $\mathcal{S}^n$ be its nth extension. Then*

$$H_r(\mathcal{S}) \leq \frac{\text{MinAveCodeLen}_r(\mathcal{S}^n)}{n} < H_r(\mathcal{S}) + \frac{1}{n}$$

where $\text{MinAveCodeLen}_r(\mathcal{S}^n)$ is the minimum average codeword length among all uniquely decipherable r-ary encoding schemes for $\mathcal{S}^n$. □

Since each codeword in the nth extension $\mathcal{S}^n$ encodes n source symbols from $\mathcal{S}$, the number

$$\frac{\text{MinAveCodeLen}_r(\mathcal{S}^n)}{n}$$

is the minimum average codeword length per source symbol of $\mathcal{S}$, taken over all uniquely decipherable r-ary encodings of $\mathcal{S}^n$. Furthermore, since $1/n$ tends to 0 as n gets large, the upper bound $H_r(\mathcal{S}) + 1/n$ approaches the lower bound $H_r(\mathcal{S})$, and so, according to the Noiseless Coding Theorem,

the number $\text{MinAveCodeLen}_r(\mathcal{S}^n)/n$ can be made as close to $H_r(\mathcal{S}^n)$ as desired by taking n large enough.

In other words, by encoding extensions of $\mathcal{S}$, that is, blocks of source symbols rather than individual source symbols, we can reduce the average codeword length per source symbol to as close to the entropy $H_r(\mathcal{S})$ as desired. This is the real essence of the Noiseless Coding Theorem. The penalty for doing so is that, since $|S^n| = q^n$, the number of codewords required to encode the nth extension $\mathcal{S}^n$ grows exceedingly large as n gets large. As a result, achieving the desired "closeness" to the entropy may be a practical impossibility.

Exercises

1. Consider the source $\mathcal{S} = (S, \mathcal{P})$, where $S = \{a, b, c\}$ and $\mathcal{P}(a) = 1/2$, $\mathcal{P}(b) = 1/4$, $\mathcal{P}(c) = 1/4$. What is the binary entropy of this source? Can we achieve a minimum average codeword length equal to the entropy for this source?
2. Consider the source $\mathcal{S} = (S, \mathcal{P})$, where $S = \{a, b, c\}$ and $\mathcal{P}(a) = 2/3$, $\mathcal{P}(b) = 1/6$, $\mathcal{P}(c) = 1/6$. What is the binary entropy of this source? Can we achieve a minimum average codeword length equal to the entropy for this source?
3. Let $\mathcal{S}$ be a binary source (thus $S = \{0, 1\}$). In order to guarantee that the average codeword length, per source symbol of $\mathcal{S}$, is at most 0.01 greater than the entropy of $\mathcal{S}$, which extension of $\mathcal{S}$ should we encode? How many codewords would we need?
4. In this exercise, we construct an entirely different type of encoding scheme. Rather than encoding each source symbol with a fixed codeword (as in Huffman encoding), source symbols are encoded in groups in a way that depends on each symbol's relationship to other source symbols in the source message. To be specific, let the source alphabet be $S = \{0, 1\}$ and suppose that $\mathcal{P}(0) = p$ and $\mathcal{P}(1) = 1 - p$. To encode a string of source symbols, we count the number of 0s occurring in the string before the appearance of a 1. The two encoding rules are

   (a) if eight 0s appear in a row, encode these 0's as a 0, that is,

   $$00000000 \rightarrow 0$$

(b) if fewer than eight 0s appear (say k 0s) before the next 1, then determine the 3-bit binary representation of k (say $e_1e_2e_3$) and encode the string of k 0s followed by the 1 as the codeword $1e_1e_2e_3$. For instance, the source string 0001 (which is three 0s followed by a 1) is encoded as 1011 since 011 is the binary representation of the number three.

i. Show that the resulting code is instantaneous.

ii. What is the probability that the source will emit k 0s followed by a 1? (Recall that we assume independence of the source emissions.)

iii. Define an event as the construction of a codeword. Find the average codeword length per event.

iv. Find the average number of source bits per event.

v. For each event, compute the number of codeword bits needed per source bit. Then compute the average of these numbers.

vi. For $p = 0.9$, determine the average codeword length per source bit for a Huffman encoding of the fourth extension $\mathcal{S}^4$. How does this number compare to the number in part v)? What significance does this have for the optimality of Huffman encoding? Does this violate the Noiseless Coding Theorem?

5. Let $\mathcal{S}$ be a source and let $\mathcal{S}^2$ be its second extension. Is the second extension of $\mathcal{S}^2$ equal to the fourth extension of $\mathcal{S}$? In symbols, is $(\mathcal{S}^2)^2 = \mathcal{S}^4$?