7

Codes for Integers

This chapter is an aside, which may safely be skipped.

Solution to exercise 6.22 (p.127)

To discuss the coding of integers we need some definitions.

The standard binary representation of a positive integer n will be denoted by $c_{\rm b}(n)$, e.g., $c_{\rm b}(5) = \texttt{101}$, $c_{\rm b}(45) = \texttt{101101}$.

The standard binary length of a positive integer n, $l_{\rm b}(n)$, is the length of the string $c_{\rm b}(n)$. For example, $l_{\rm b}(5) = 3$, $l_{\rm b}(45) = 6$.

The standard binary representation $c_{\rm b}(n)$ is *not* a uniquely decodeable code for integers since there is no way of knowing when an integer has ended. For example, $c_{\rm b}(5)c_{\rm b}(5)$ is identical to $c_{\rm b}(45)$. It would be uniquely decodeable if we knew the standard binary length of each integer before it was received.

Noticing that all positive integers have a standard binary representation that starts with a `1`, we might define another representation:

The headless binary representation of a positive integer n will be denoted by $c_{\rm B}(n)$, e.g., $c_{\rm B}(5) = \texttt{01}$, $c_{\rm B}(45) = \texttt{01101}$ and $c_{\rm B}(1) = \lambda$ (where λ denotes the null string).

This representation would be uniquely decodeable if we knew the length $l_{\rm b}(n)$ of the integer.

So, how can we make a uniquely decodeable code for integers? Two strategies can be distinguished.

1. **Self-delimiting codes**. We first communicate somehow the length of the integer, $l_{\rm b}(n)$, which is also a positive integer; then communicate the original integer n itself using $c_{\rm B}(n)$.

2. **Codes with 'end of file' characters**. We code the integer into blocks of length b bits, and reserve one of the 2^b symbols to have the special meaning 'end of file'. The coding of integers into blocks is arranged so that this reserved symbol is not needed for any other purpose.

The simplest uniquely decodeable code for integers is the unary code, which can be viewed as a code with an end of file character.

Unary code. An integer n is encoded by sending a string of $n-1$ 0s followed by a 1.

n	$c_{\mathrm{U}}(n)$
1	1
2	01
3	001
4	0001
5	00001
⋮	
45	000000000000000000000000000000000000000000001

The unary code has length $l_{\mathrm{U}}(n) = n$.

The unary code is the optimal code for integers if the probability distribution over n is $p_{\mathrm{U}}(n) = 2^{-n}$.

Self-delimiting codes

We can use the unary code to encode the *length* of the binary encoding of n and make a self-delimiting code:

Code C_α. We send the unary code for $l_{\mathrm{b}}(n)$, followed by the headless binary representation of n.

$$c_\alpha(n) = c_{\mathrm{U}}[l_{\mathrm{b}}(n)]c_{\mathrm{B}}(n). \tag{7.1}$$

Table 7.1 shows the codes for some integers. The overlining indicates the division of each string into the parts $c_{\mathrm{U}}[l_{\mathrm{b}}(n)]$ and $c_{\mathrm{B}}(n)$. We might equivalently view $c_\alpha(n)$ as consisting of a string of $(l_{\mathrm{b}}(n) - 1)$ zeroes followed by the standard binary representation of n, $c_{\mathrm{b}}(n)$.

The codeword $c_\alpha(n)$ has length $l_\alpha(n) = 2l_{\mathrm{b}}(n) - 1$.

The implicit probability distribution over n for the code C_α is separable into the product of a probability distribution over the length l,

$$P(l) = 2^{-l}, \tag{7.2}$$

and a uniform distribution over integers having that length,

$$P(n \mid l) = \begin{cases} 2^{-l+1} & l_{\mathrm{b}}(n) = l \\ 0 & \text{otherwise.} \end{cases} \tag{7.3}$$

n	$c_{\mathrm{b}}(n)$	$l_{\mathrm{b}}(n)$	$c_\alpha(n)$
1	1	1	$\overline{1}$
2	10	2	$\overline{01}0$
3	11	2	$\overline{01}1$
4	100	3	$\overline{001}00$
5	101	3	$\overline{001}01$
6	110	3	$\overline{001}10$
⋮			
45	101101	6	$\overline{00000}101101$

Table 7.1. C_α.

Now, for the above code, the header that communicates the length always occupies the same number of bits as the standard binary representation of the integer (give or take one). If we are expecting to encounter large integers (large files) then this representation seems suboptimal, since it leads to all files occupying a size that is double their original uncoded size. Instead of using the unary code to encode the length $l_{\mathrm{b}}(n)$, we could use C_α.

Code C_β. We send the length $l_{\mathrm{b}}(n)$ using C_α, followed by the headless binary representation of n.

$$c_\beta(n) = c_\alpha[l_{\mathrm{b}}(n)]c_{\mathrm{B}}(n). \tag{7.4}$$

Iterating this procedure, we can define a sequence of codes.

Code C_γ.

$$c_\gamma(n) = c_\beta[l_{\mathrm{b}}(n)]c_{\mathrm{B}}(n). \tag{7.5}$$

Code C_δ.

$$c_\delta(n) = c_\gamma[l_{\mathrm{b}}(n)]c_{\mathrm{B}}(n). \tag{7.6}$$

n	$c_\beta(n)$	$c_\gamma(n)$
1	$\overline{1}$	$\overline{1}$
2	$\overline{010}0$	$\overline{0100}0$
3	$\overline{010}1$	$\overline{0100}1$
4	$\overline{011}00$	$\overline{0101}00$
5	$\overline{011}01$	$\overline{0101}01$
6	$\overline{011}10$	$\overline{0101}10$
⋮		
45	$\overline{0011001}101$	$\overline{0111001}101$

Table 7.2. C_β and C_γ.

Codes with end-of-file symbols

We can also make byte-based representations. (Let's use the term byte flexibly here, to denote any fixed-length string of bits, not just a string of length 8 bits.) If we encode the number in some base, for example decimal, then we can represent each digit in a byte. In order to represent a digit from 0 to 9 in a byte we need four bits. Because $2^4 = 16$, this leaves 6 extra four-bit symbols, {1010, 1011, 1100, 1101, 1110, 1111}, that correspond to no decimal digit. We can use these as end-of-file symbols to indicate the end of our positive integer.

Clearly it is redundant to have more than one end-of-file symbol, so a more efficient code would encode the integer into base 15, and use just the sixteenth symbol, 1111, as the punctuation character. Generalizing this idea, we can make similar byte-based codes for integers in bases 3 and 7, and in any base of the form $2^n - 1$.

These codes are almost complete. (Recall that a code is 'complete' if it satisfies the Kraft inequality with equality.) The codes' remaining inefficiency is that they provide the ability to encode the integer zero and the empty string, neither of which was required.

n	$c_3(n)$	$c_7(n)$
1	01 11	001 111
2	10 11	010 111
3	01 00 11	011 111
⋮		
45	01 10 00 00 11	110 011 111

Table 7.3. Two codes with end-of-file symbols, C_3 and C_7. Spaces have been included to show the byte boundaries.

▷ Exercise 7.1.[2, p.136] Consider the implicit probability distribution over integers corresponding to the code with an end-of-file character.

(a) If the code has eight-bit blocks (i.e., the integer is coded in base 255), what is the mean length in bits of the integer, under the implicit distribution?

(b) If one wishes to encode binary files of expected size about one hundred kilobytes using a code with an end-of-file character, what is the optimal block size?

Encoding a tiny file

To illustrate the codes we have discussed, we now use each code to encode a small file consisting of just 14 characters,

Claude Shannon.

- If we map the ASCII characters onto seven-bit symbols (e.g., in decimal, C = 67, l = 108, etc.), this 14 character file corresponds to the integer

$$n = 167\,987\,786\,364\,950\,891\,085\,602\,469\,870 \quad \text{(decimal)}.$$

- The unary code for n consists of this many (less one) zeroes, followed by a one. If all the oceans were turned into ink, and if we wrote a hundred bits with every cubic millimeter, there might be enough ink to write $c_{\mathrm{U}}(n)$.

- The standard binary representation of n is this length-98 sequence of bits:

$$c_{\mathrm{b}}(n) = \begin{array}{l} 10000111101100110000111101011100100110010101000000 \\ 10100111101000110000111011101101110111011111101110. \end{array}$$

▷ Exercise 7.2.[2] Write down or describe the following self-delimiting representations of the above number n: $c_\alpha(n)$, $c_\beta(n)$, $c_\gamma(n)$, $c_\delta(n)$, $c_3(n)$, $c_7(n)$, and $c_{15}(n)$. Which of these encodings is the shortest? [Answer: c_{15}.]

Comparing the codes

One could answer the question 'which of two codes is superior?' by a sentence of the form 'For $n > k$, code 1 is superior, for $n < k$, code 2 is superior' but I contend that such an answer misses the point: any complete code corresponds to a prior for which it is optimal; you should not say that any other code is superior to it. Other codes are optimal for other priors. These implicit priors should be thought about so as to achieve the best code for one's application.

Notice that one cannot, for free, switch from one code to another, choosing whichever is shorter. If one were to do this, then it would be necessary to lengthen the message in some way that indicates which of the two codes is being used. If this is done by a single leading bit, it will be found that the resulting code is suboptimal because it fails the Kraft equality, as was discussed in exercise 5.33 (p.104).

Another way to compare codes for integers is to consider a sequence of probability distributions, such as monotonic probability distributions over $n \geq 1$, and rank the codes as to how well they encode *any* of these distributions. A code is called a 'universal' code if for any distribution in a given class, it encodes into an average length that is within some factor of the ideal average length.

Let me say this again. We are meeting an alternative world view – rather than figuring out a good prior over integers, as advocated above, many theorists have studied the problem of creating codes that are reasonably good codes for *any* priors in a broad class. Here the class of priors conventionally considered is the set of priors that (a) assign a monotonically decreasing probability over integers and (b) have finite entropy.

Several of the codes we have discussed above are universal. Another code which elegantly transcends the sequence of self-delimiting codes is Elias's 'universal code for integers' (Elias, 1975), which effectively chooses from all the codes $C_\alpha, C_\beta, \ldots$. It works by sending a sequence of messages each of which encodes the length of the next message, and indicates by a single bit whether or not that message is the final integer (in its standard binary representation). Because a length is a positive integer and all positive integers begin with '1', all the leading 1s can be omitted.

Algorithm 7.4. Elias's encoder for an integer n.

```
Write '0'
Loop {
   If ⌊log n⌋ = 0 halt
   Prepend c_b(n) to the written string
   n:=⌊log n⌋
}
```

The encoder of C_ω is shown in algorithm 7.4. The encoding is generated from right to left. Table 7.5 shows the resulting codewords.

▷ Exercise 7.3.[2] Show that the Elias code is not actually the best code for a prior distribution that expects very large integers. (Do this by constructing another code and specifying how large n must be for your code to give a shorter length than Elias's.)

n	$c_\omega(n)$	n	$c_\omega(n)$	n	$c_\omega(n)$	n	$c_\omega(n)$
1	0	9	1110010	31	10100111110	256	1110001000000000
2	100	10	1110100	32	101011000000	365	1110001011011010
3	110	11	1110110	45	101011011010	511	1110001111111110
4	101000	12	1111000	63	101011111110	512	11100110000000000
5	101010	13	1111010	64	1011010000000	719	11100110110011110
6	101100	14	1111100	127	1011011111110	1023	11100111111111110
7	101110	15	1111110	128	10111100000000	1024	111010100000000000
8	1110000	16	10100100000	255	10111111111110	1025	111010100000000010

Table 7.5. Elias's 'universal' code for integers. Examples from 1 to 1025.

Solutions

Solution to exercise 7.1 (p.134). The use of the end-of-file symbol in a code that represents the integer in some base q corresponds to a belief that there is a probability of $(1/(q+1))$ that the current character is the last character of the number. Thus the prior to which this code is matched puts an exponential prior distribution over the length of the integer.

(a) The expected number of characters is $q+1 = 256$, so the expected length of the integer is $256 \times 8 \simeq 2000$ bits.

(b) We wish to find q such that $q \log q \simeq 800\,000$ bits. A value of q between 2^{15} and 2^{16} satisfies this constraint, so 16-bit blocks are roughly the optimal size, assuming there is one end-of-file character.