Concepts in Data Compression

In this section the basic concepts of data compression are shown as below:

Entropy

Entropy is a concept in thermodynamics, statistical mechanics and information theory. Both concepts of entropy have deep links with one another, although it took many years for the development of the theories of statistical mechanics and information theory to make this connection apparent. This section is about information entropy, the information-theoretic formulation of entropy. Information entropy is occasionally called Shannon's entropy in honor of Claude E. Shannon, who formulated many of the key ideas of information theory. Claud Shannon's paper "A mathematical theory of communication" published in July and October of 1948 is the Magna Carta of the information age. Shannon's discovery of the fundamental laws of data compression and transmission marks the birth of Information Theory. The concept of entropy in information theory describes how much information there is in a signal or event.

An intuitive understanding of information entropy relates to the amount of uncertainty about an event associated with a given probability distribution. As an example, consider a box containing many colored balls. If the balls are all of different colors and no color predominates, then our uncertainty about the color of a randomly drawn ball is maximal. On the other hand, if the box contains more red balls than any other color, then there is slightly less uncertainty about the result: the ball drawn from the box has more chances of being red (if we were forced to place a bet, we would bet on a red ball). Telling someone the color of every new drawn ball provides them with more information in the first case than it does in the second case, because there is more uncertainty about what might happen in the first case than there is in the second. Intuitively, if we know the number of balls remaining, and they are all of one color, then there is no uncertainty about what the next ball drawn will be, and therefore there is no information content from drawing the ball. As a result, the entropy of the "signal" (the sequence of balls drawn, as calculated from the probability distribution) is higher in the first case than in the second.

For a set of possible messages, Shannon defined entropy as,

$$H(S) = \sum_{s \in S} p(s) \log_2 \frac{1}{p(s)}. \qquad (1) \quad \text{OR}$$

$$H = -\sum_{i=0}^{n-1} P_i \log_2 P_i$$

Where p(s) is the probability of message s. The definition of Entropy is very similar to that in statistical physics- in physics S is the set of possible states a system can be in and p(s) is the probability the system is in state (s). We might remember that the second law of thermodynamics basically says that the entropy of a system and its surroundings can only increase. Getting back to messages, if we consider the individual messages $s \in S$, Shannon defined the notion of theself-information of a message as

$$i(s) = \log_2 \frac{1}{p(s)}. \qquad (2)$$

This self-information represents the number of bits of information contained in it and, roughly speaking, the number of bits we should use to send that message. The equation says that messages with higher probability will contain less information.

The entropy is simply a weighted average of the information of each message, and therefore the average number of bits of information in the set of messages. Larger entropies represent more information. Here are some examples of entropies for different probability distributions over five messages:

$$p(S) = \{0.25, 0.25, 0.25, 0.125, 0.125\}$$
$$H = 3 \times 0.25 \times \log_2 4 + 2 \times 0.125 \times \log_2 8$$
$$= 1.5 + 0.75$$
$$= 2.25$$
$$p(s) = \{0.75, 0.625, 0.625, 0.625, 0.625\}$$
$$H = 0.75 \times \log_2 \frac{4}{3} + 4 \times 0.625 \times \log_2 16$$
$$= 0.3 + 1$$
$$= 1.3$$

The Unary Code

The unary code of the non-negative integer n is defined as n-1 ones followed by one zero or, alternatively, as n-1 zeros followed by a single one.

Table: Some Unary Codes

N	Code	Alt. Code
1	0	1
2	10	01
3	110	001
4	1110	0001
5	11110	00001

Ad Hoc Text Compression

Here are some simple, intuitive ideas for cases where the compression must be reversible (lossless). If the text contains many spaces but they are not clustered, they may be removed, and their positions indicated by a bit-string that contains a 0 for each text character that is not a space and a 1 for each space. Thus, the text

Here are some ideas,

Is encoded as the bit-string "0000100010000100000" followed by the text

Herearesomeideas.

Variable and Fixed Length Codes

Variable length codes are desirable for data compression because overall savings may be achieved by assigning short codewords to frequently occurring symbols and long codewords to rarely occurring ones. For example, consider a variable length code (0, 100, 101, 110, 111) with lengths of codewords (1, 3, 3, 3, 3) for alphabet (A, B, C, D, E), and a source string BAAAAAAAC with frequencies for each symbol (7, 1, 1, 0, 0). The average number of bits required is

$$\bar{l} = \frac{1 \times 7 + 3 \times 1 + 3 \times 1}{9} \approx 1.4 \text{ bits/symbol}$$

This is almost a saving of half the number of bits compared to 3 bits/symbol using a 3-bit fixed length code. The shorter the codewords, the shorter the total length of a source file. Hence the code would be a better one from the compression point of view.

Uniquely Decodable Codes

The average length of the code is not the only important point in designing a "good" code. Consider the following example. Suppose our source alphabet consists of four letters a1, a2, a3, and a4, with probabilities P(a1) = 1/2, P(a2) = 1/4, and P(a3) = P(a4) = 1/8. The entropy for this source is 1.75 bits/symbol. Consider the codes for this source in Table 2.2.

TABLE 2.2 Four different codes for a four-letter alphabet.

Letters	Probability	Code 1	Code 2	Code 3	Code 4
a_1	0.5	0	0	0	0
a_2	0.25	0	1	10	01
a_3	0.125	1	00	110	011
a_4	0.125	10	11	111	0111
Average length		1.125	1.25	1.75	1.875

The average length ***l*** for each code is given by

$$l = \sum_{i+1}^{*} P(a_i)n(a_i)$$

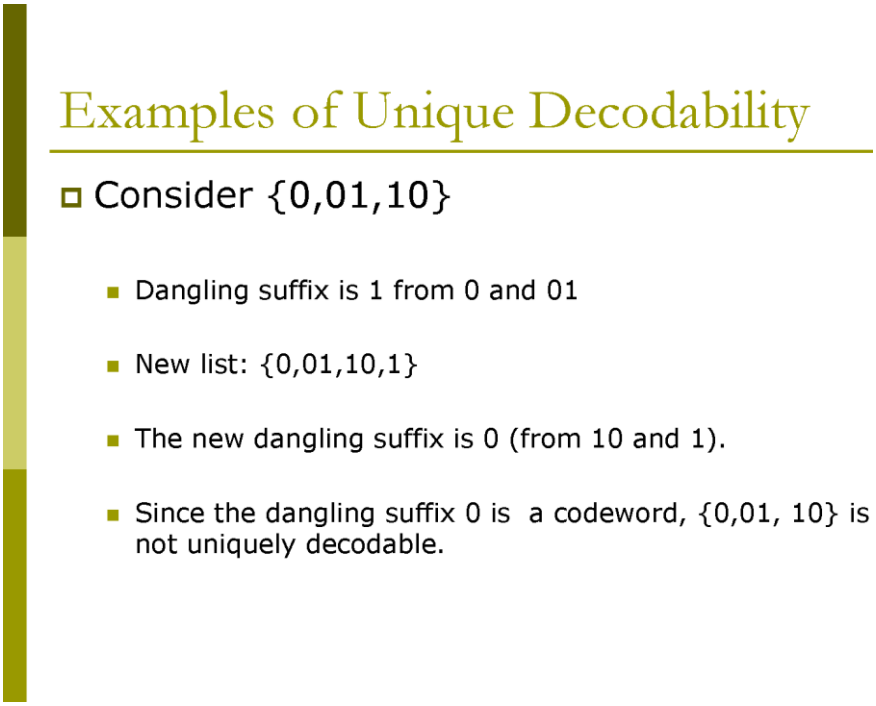
Where n (a_i) is the number of bits in the codeword for letter $\mathbf{a_i}$ and the average length is given in bits/symbol. Based on the average length, Code 1 appears to be the best code. However, to be useful, a code should have the ability to transfer information in an unambiguous manner. This is obviously not the case with Code 1. Both a1 and a2 have been assigned the codeword 0. When a 0 is received, there is no way to know whether an a1 was transmitted or an a2. We would like each symbol to be assigned a unique codeword. At first glance, Code 2 does not seem to have the problem of ambiguity; each symbol is assigned a distinct codeword. However, suppose we want to encode the sequence a2 a1 a1. Using Code 2, we would encode this with the binary string 100. However, when the string 100 is received at the decoder, there are several ways in which the decoder can decode this string. The string 100 can be decoded as a2a1a1, or as a2a3. This means that once a sequence is encoded with Code 2, the original sequence cannot be recovered with certainty.

How do we know a uniquely decodable code?

- Consider two codewords: 011 and 011101
  - Prefix: 011
  - Dangling suffix: 101
- Algorithm:
  1. Construct a list of all the codewords.
  2. Examine all pairs of codewords to see if any codeword is a prefix of another codeword. If there exists such a pair, add the dangling suffice to the list unless there is one already.
  3. Continue this procedure using the larger list until:
     1. Either a dangling suffix is a codeword -> not uniquely decodable.
     2. There are no more unique dangling suffixes -> uniquely decodable.

Examples of Unique Decodability

- Consider {0,01,11}
  - Dangling suffix is 1 from 0 and 01
  - New list: {0,01,11,1}
  - Dangling suffix is 1 (from 0 and 01, and also 1 and 11), and is already included in previous iteration.
  - Since the dangling suffix is not a codeword, {0,01, 11} is uniquely decodable.



Prefix Codes and Binary Trees

A prefix is the first few consecutive bits of a codeword. When two codewords are of different lengths, it is possible that the shorter codeword is identical to the first few bits of the longer codeword. In this case, the shorter codeword is said to be a prefix of the longer one. **A prefix code is a uniquely decodable code: given a complete and accurate sequence, a receiver can identify each word without requiring a special marker between words. However, there are uniquely decodable codes that are not prefix codes.**

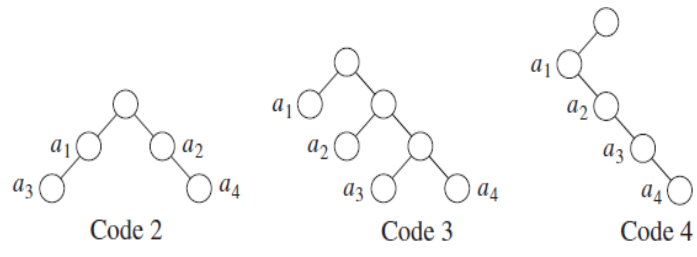
Example: Consider two binary codewords of different length: c1 = 010 (3 bits) and c2 = 01011 (5 bits). The shorter codeword c1 is the prefix of the longer code c2 as c2= 01011. Codeword c2 can be obtained by appending two more bits 11 to cl.

The prefix property of a binary code is the fact that no codeword is a prefix of another. A prefix code is a code in which no codeword is a prefix of another codeword. It is easy to check whether a binary code is a prefix code by drawing an associated binary tree. Each binary code can correspond to one such binary tree, in which each codeword corresponds to a path from the root to a node with the codeword name marked at the end of the path. Each bit 0 in a codeword corresponds to a left edge and each 1 to a right edge. Recall that, if a prefix code is represented in such an associate binary tree, all the codeword labels will be at its leaves. Two steps are involved in this approach:

1. **Construct Binary Tree**

First, we create a node as the root of the binary tree. Next, we look at the codewords one by one. For each codeword, we read one bit at a time from the first to the last. Starting from the root, we either draw a new branch or move down an edge along a branch according to the value of the bit. *When a bit 0 is read*, we draw, if there is no branch yet, a left branch and a new node at the end of the branch. We move down one edge along the left branch otherwise and arrive at the node at the end of the edge. *Similarly, when a bit 1* is read, we draw if there no branch yet, a right branch, or move down an edge along the right branch otherwise. The process repeats from node to node while reading the bit by bit until the end of the codeword. We mark the codeword after finishing with the whole codeword.

Example: Draw the binary tree for Code 2, Code 3, and Code 4 in Table 2.2.



Binary trees for three different codes.

2. **Checking Codeword Position**

If all the codeword labels are only associated with the leaves, thenthe codeword is a prefix code. Otherwise, it is not.

Example: Decide whether the codes (1,01,001,0000) and (0,10,110,1011) for alphabet (A,B,C,D) are prefix codes. Figure below shows the solution.

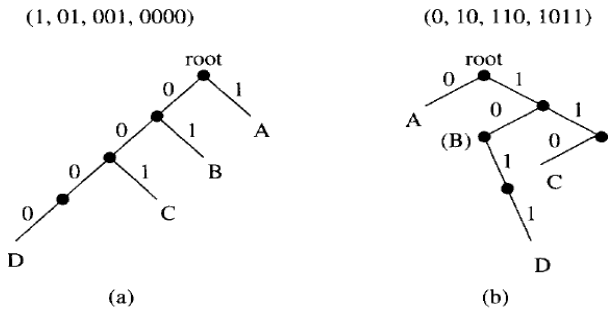


Figure 2.6: Prefix property and binary trees

Decoding Prefix Codeword