Chapter 3

Compression

In Chapter 1 we examined the fundamental unit of information, the bit, and its various abstract representations: the Boolean bit, the circuit bit, the control bit, the physical bit, the quantum bit, and the classical bit. Our never-ending quest for improvement made us want representations of single bits that are smaller, faster, stronger, smarter, safer, and cheaper.

In Chapter 2 we considered some of the issues surrounding the representation of complex objects by arrays of bits (at this point, Boolean bits). The relationship between the objects to be represented (the symbols) and the corresponding arrays of bits (known as *codewords*) is known as a *code*. We naturally want codes that are smaller and stronger, i.e., that use shorter codewords and are less vulnerable to errors. In this chapter we will consider how to get smaller codes. In Chapter 4 we will look at techniques of avoiding errors.

In Chapter 2 we considered systems of the sort shown in Figure 3.1, in which symbols are encoded into bit strings, which are transported (in space and/or time) to a decoder, which then recreates the original symbols.

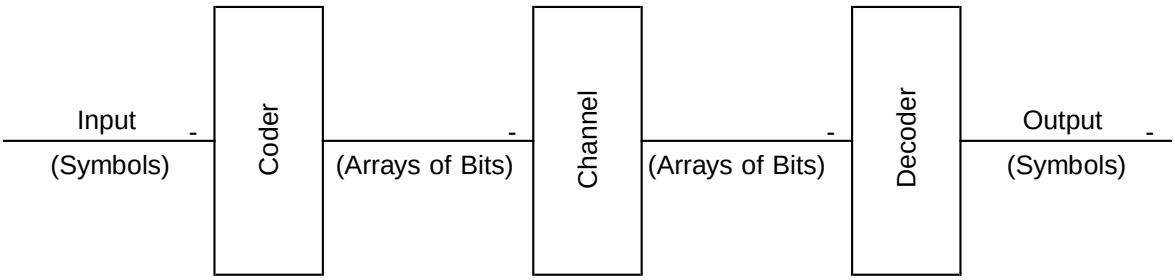

Figure 3.1: Generalized communication system

Typically the same code is used for a sequence of symbols, one after another. Such a sequence forms a message. The role of data compression is to convert the string of bits representing a message into a shorter string for more economical transmission, storage, or processing. The result is the system in Figure 3.2, with a *compressor* and an *expander* added. Ideally, the expander should exactly reverse the action of the compressor so that the coder and decoder need not be changed.

On first thought, this approach might seem surprising. Why is there any reason to believe that the same

Author: Paul Penfield, Jr.
This document: http://www.mtl.mit.edu/Courses/6.050/2013/notes/chapter3.pdf
Version 1.7, February 8, 2013. Copyright © 2013 Massachusetts Institute of Technology

This algorithm is shown in program format in Figure 3.6.

```
 0  # decoding algorithm
 1  for each received code until stop code {
 2    if code is start code {
 3      reset dictionary
 4      get next code
 5      get string from dictionary  # it will be a single character
 6    }
 7    else {
 8      get string from dictionary
 9      update last dictionary entry by appending first character of string
10    }
11    add string as new dictionary entry
12    output string
13  }
```

Figure 3.6: LZW decoding algorithm

Note that the coder and decoder each create the dictionary on the fly; the dictionary therefore does not have to be explicitly transmitted, and the coder deals with the text in a single pass.

Does this work, i.e., is the number of bits needed for transmission reduced? We sent 18 8-bit characters (144 bits) in 14 9-bit transmissions (126 bits), a savings of 12.5%, for this very short example. For typical text there is not much reduction for strings under 500 bytes. Larger text files are often compressed by a factor of 2, and drawings even more.

3.7.2 LZW Algorithm, Example 2

Encode and decode the text message

`itty bitty nitty grrritty bit bin`

(again, this peculiar phrase was designed to have repeated strings so that the dictionary forms rapidly; it also has a three-long sequence `rrr` which illustrates one aspect of this algorithm).

The initial set of dictionary entries include the characters in Table 3.3, which are found in the string, along with control characters for start and stop.

32	space	114	r
98	b	116	t
103	g	121	y
105	i	256	start
110	n	257	stop

Table 3.3: LZW Example 2 Starting Dictionary

The same algorithms used in Example 1 can be applied here. The result is shown in Table 3.4. Note that the dictionary builds up quite rapidly, and there is one instance of a four-character dictionary entry transmitted. Was this compression effective? Definitely. A total of 33 8-bit characters (264 bits) were sent in 22 9-bit transmissions (198 bits, even including the start and stop characters), for a saving of 25% in bits.

There is one place in this example where the decoder needs to do something unusual. Normally, on receipt of a transmitted codeword, the decoder can look up its string in the dictionary and then output it and use its first character to complete the partially formed last dictionary entry, and then start the next dictionary entry. Thus only one dictionary lookup is needed. However, the algorithm presented above uses

two lookups, one for the first character, and a later one for the entire string. Why not use just one lookup for greater efficiency?

There is a special case illustrated by the transmission of code 271 in this example, where the string corresponding to the received code is not complete. The first character can be found but then before the entire string is retrieved, the entry must be completed. This happens when a character or a string appears for the first time three times in a row, and is therefore rare. The algorithm above works correctly, at a cost of an extra lookup that is seldom needed and may slow the algorithm down. A faster algorithm with a single dictionary lookup works reliably only if it detects this situation and treats it as a special case.

Encoding				Transmission		Decoding		
Input		New dictionary entry				New dictionary entry		Output
105	i	–	–	256	(start)	–	–	–
116	t	258	i t	105	i	–	–	i
116	t	259	t t	116	t	258	i t	t
121	y	260	t y	116	t	259	t t	t
32	space	261	y space	121	y	260	t y	y
98	b	262	space b	32	space	261	y space	space
105	i	263	b i	98	b	262	space b	b
116	t	–	–	–	–	–	–	–
116	t	264	i t t	258	i t	263	b i	i t
121	y	–	–	–	–	–	–	–
32	space	265	t y space	260	t y	264	i t t	t y
110	n	266	space n	32	space	265	t y space	space
105	i	267	n i	110	n	266	space n	n
116	t	–	–	–	–	–	–	–
116	t	–	–	–	–	–	–	–
121	y	268	i t t y	264	i t t	267	n i	i t t
32	space	–	–	–	–	–	–	–
103	g	269	y space g	261	y space	268	i t t y	y space
114	r	270	g r	103	g	269	y space g	g
114	r	271	r r	114	r	270	g r	r
114	r	–	–	–	–	–	–	–
105	i	272	r r i	271	r r	271	r r	r r
116	t	–	–	–	–	–	–	–
116	t	–	–	–	–	–	–	–
121	y	–	–	–	–	–	–	–
32	space	273	i t t y space	268	i t t y	272	r r i	i t t y
98	b	–	–	–	–	–	–	–
105	i	274	space b i	262	space b	273	i t t y space	space b
116	t	–	–	–	–	–	–	–
32	space	275	i t space	258	i t	274	space b i	it
98	b	–	–	–	–	–	–	–
105	i	–	–	–	–	–	–	–
110	n	276	space b i n	274	space b i	275	i t space	space b i
–	–	–	–	110	n	276	space b i n	n
–	–	–	–	257	(stop)	–	–	–

Table 3.4: LZW Example 2 Transmission Summary