Part I:
Basic Concepts

Our everyday experience suggests that compression is an option that we naturally select when faced with problems of high cost or restricted space. The following points illustrate how such problems have been solved throughout history by resorting to (often intuitive) compression:

- In ancient Greece, manuscripts were written on papyrus, which was then very expensive. As a result, writers tried to squeeze more text in a given space by eliminating punctuations and interword spaces (Figure 1).
- In ancient Rome, people went around the high cost of tombstone engraving by resorting to acronyms, the most common of which were `S.T.L.` (Sit Terra Levit, or let the earth rest lightly upon her), `D.M.` (Dis Manibus, or to the ghosts of the underworld), and `B.M.` (Bene Merenti, or to one deserving well).

Figure 1: Greek Papyrus and Ancient Coin.

- In the middle ages, praise for the current ruler was often stamped onto coins in the form of acronyms, because of the tight space available (Figure 1).

- In a natural language, common words tend to be shorter than rarely-used words. It is hard to imagine a language where the word for, say, `yes` would be as long as `encyclopedia` or the word for `establishment` would be as short as `me`.

- We are familiar with the term "fine print." This term is often used to hide unfriendly clauses in a contract, or negative aspects of an item being advertised. Sometimes, however, small print is simply used to save space, as is common in newspapers.

- The Arabic numerals that we use are based on weights assigned to positions in the number. Thus, the digit 4 in 24,806 has a weight of $10^3 = 1000$, so its value is 4,000. This numbering system has many advantages, not the least of which is that the numbers are short. They are shorter than Roman numerals and much shorter than stone-age numerals (see the discussion of stone-age binary in Section 1.1.1).

- The well-known Morse code (Section 1.1) assigns short codes to common letters, such as `E` and `T` and long codes to rare letters, such as `Z` and `Q`. This is an early example of intuitive text compression.

- The six-shutter telecommunication system, used by the British admiralty in the 19th century, could transmit 64 different symbols, more than enough for the letters and digits. The extra symbols were assigned to common words. This system is described in [Holzmann and Pehrson 95] and its application to compression is mentioned in [Bell et al. 90].

- A similar system is the well-known Braille code for the blind. Developed by Louis Braille in the 1820s, this code consists of groups (or cells) of 3×2 dots each, embossed on thick paper. Each of the six dots in a group may be flat or raised, implying that the information content of a group is equivalent to six bits, resulting in 64 possible groups. Once appropriate codes are assigned to the letters, digits, and common punctuation marks, several groups remain and may be used to code common words—such as `and`, `for`, and `of`—and common strings of letters—such as `ound`, `ation`, and `th`.

- Scientists often claim that the chief aim of science is to explain as many known facts as possible by deduction from as few assumptions (or axioms) as possible. This is an example of economy of expression.

- Similarly, Occam's razor (attributed to the 14th-century logician William of Ockham) is a principle which states that the explanation of a phenomenon should make as few assumptions as possible (entia non sunt multiplicanda praeter necessitatem, or entities should not be multiplied beyond necessity).

The first part of this book consists of the first three chapters. They discuss the basic approaches to data compression and describe a few popular techniques and methods that are commonly used to compress data. Chapter 1 introduces the reader to the important concepts of variable-length codes, prefix codes, statistical distributions, run-length encoding, dictionary compression, transforms, and quantization. Chapter 2 is

devoted to the important Huffman algorithm and codes, and Chapter 3 describes some of the many dictionary-based compression methods.

There are four basic food groups: milk chocolate, dark chocolate, white chocolate, and chocolate truffles.

—Anonymous

Introduction

The modern discipline of data compression is concerned with reducing the size of digital binary data. A data compression algorithm inputs a bitstream (a disk file or bits read from a network) and outputs a shorter bitstream. Most of the physical objects surrounding us are difficult or impossible to shrink (or are damaged when forcibly compressed), so shrinking the size of a data file may seem like magic (or perhaps like cheating). Thus, before we try to explain *how* data is compressed, it is important to explain *why* it can be compressed. The key to compressing data is the distinction between data and information. Data is how information is represented; it is the physical embodiment of the information. We know that it is possible to use different amounts of data to convey the same information. A good example is a story. A novel that originally occupies 300 pages can be "digested" and compressed to just 30 pages without losing the main outlines of the plot. The same story may be told by one person in 2000 words and by another in 200 words because the former employs unnecessary (or irrelevant) words, thus introducing redundancy into his narrative, while the latter selects only those words that are strictly needed.

In simple terms, data can be compressed because its original representation is not the shortest possible. We use different digital data structures to represent various types of information in our computers, and we use these particular structures because they make it easy to visualize the information and operate on it. Compression changes the data representation to a shorter one (ideally, the shortest one), but it is difficult or even impossible to visualize and process the information in such a representation.

In technical terms we say that the original representation of data has redundancies and compressing the data reduces or eliminates these redundancies. Random data is just that, random; it has no structure. Any nonrandom data is nonrandom because it has structure in the form of regular patterns, and it is this structure that introduces redundancies into the data. Data that has no redundancy to begin with cannot be compressed. Thus, compression of data is not absolute. Given a data file, we cannot tell whether it is small enough or too large. We have to look for redundancies (in terms of structures or patterns) in the data and compress the data by eliminating them. Compression should always be measured by comparing the size of the compressed data with the size of the original data.

The interpretation of compression as the removal of redundancy also explains why it is impossible to compress data that has already been compressed. When data is compressed, any redundancies in it, in the form of structures or patterns, is removed. The compressed data features no structure and cannot be distinguished from random data; in fact, it is random. Thus, any attempt to compress it again will fail. If it were possible to compress data that is already compressed, then we could start with a data file A, compress it to a smaller file B, compress B in turn to a smaller file C, and continue in this way until a 1-byte (or even a 1-bit) file is reached. A 1-byte file cannot contain all the data of file A (whose size is arbitrary), so recursive compression is impossible.

The following simple argument illustrates the essence of the statement "Data compression is achieved by reducing or removing redundancy in the data." The argument shows that most data files cannot be compressed, no matter what compression method is used. This seems strange at first because we compress our data files all the time. The point is that most files cannot be compressed because they are random or close to random and therefore have no redundancy. The (relatively) few files that can be compressed are the ones that we *want* to compress; they are the files we use all the time. They have redundancy, are nonrandom and are therefore useful and interesting.

Here is the argument. Given two different files A and B that are compressed to files C and D, respectively, it is clear that C and D must be different. If they were identical, there would be no way to decompress them and get back file A or file B.

Suppose that a file of size n bits is given and we want to compress it efficiently. Any compression method that can compress this file to, say, 10 bits would be welcome. Even compressing it to 11 bits or 12 bits would be great. We therefore (somewhat arbitrarily) assume that compressing such a file to half its size or better is considered good compression. There are 2^n n-bit files and they would have to be compressed into 2^n different files of sizes less than or equal to $n/2$. However, the total number of these files is

$$N = 1 + 2 + 4 + \cdots + 2^{n/2} = 2^{1+n/2} - 1 \approx 2^{1+n/2},$$

so only N of the 2^n original files have a chance of being compressed efficiently. The problem is that N is much smaller than 2^n. Here are two examples of the ratio between these two numbers.

For $n = 100$ (files with just 100 bits), the total number of files is 2^{100} and the number of files that can be compressed efficiently is 2^{51}. The ratio of these numbers is the ridiculously small fraction $2^{-49} \approx 1.78 \times 10^{-15}$.

For $n = 1000$ (files with just 1000 bits, about 125 bytes), the total number of files is 2^{1000} and the number of files that can be compressed efficiently is 2^{501}. The ratio of these numbers is the incredibly small fraction $2^{-499} \approx 9.82 \times 10^{-91}$.

Most files of interest are at least some thousands of bytes long. For such files, the percentage of files that can be efficiently compressed is so small that it cannot be computed with floating-point numbers even on a supercomputer (the result comes out as zero).

The 50% figure used here is arbitrary, but even increasing it to 90% isn't going to make a significant difference. Here is why. Assuming that a file of n bits is given and that $0.9n$ is an integer, the number of files of sizes up to $0.9n$ is

$$2^0 + 2^1 + \cdots + 2^{0.9n} = 2^{1+0.9n} - 1 \approx 2^{1+0.9n}.$$

For $n = 100$, there are 2^{100} files and $2^{1+90} = 2^{91}$ can be compressed well. The ratio of these numbers is $2^{91}/2^{100} = 2^{-9} \approx 0.00195$. For $n = 1000$, the corresponding fraction is $2^{901}/2^{1000} = 2^{-99} \approx 1.578 \times 10^{-30}$. These are still extremely small fractions.

It is therefore clear that no compression method can hope to compress all files or even a significant percentage of them. In order to compress a data file, the compression algorithm has to examine the data, find redundancies in it, and try to remove them. The redundancies in data depend on the type of data (text, images, audio, etc.), which is why a new compression method has to be developed for a specific type of data and it performs best on this type. There is no such thing as a universal, efficient data compression algorithm.

In spite of the arguments above, there are always those who claim to have developed a "magic" compression method that can compress any data file to a small fraction of its original size. Reference [incredible 07] lists quite a few such claims.

Multimedia digital data. The first computers were conceived as fast, reliable computing machines, but it did not take computer users long to realize that the computer can also process nonnumeric data. The various compilers for programming languages are one such example, as are also databases, computer games, and computer networks. However, it was not until the 1990s that many *multimedia* applications were developed and came into popular use. The term "multimedia" refers to the ability to digitize, store, and manipulate in the computer all kinds of data, not just numbers. Today (2008), computer users commonly create, edit, store, view, and exchange text, still images, video, and audio data easily and reliably.

> Multimedia (noun, plural), the use of different media to convey information; text together with audio, graphics and animation, often packaged on CD-ROM with links to the Internet.
>
> —wiktionary.com

Each type of data is represented differently in the computer and features different structures and redundancies. This is why different approaches and techniques are needed to compress it. Following is a discussion of the representations and redundancies of the main data types.

Text is represented in the computer as individual characters, each encoded in binary. The codes are all the same length, because fixed-size codes are easy to store in memory, move about, and operate on. For many years, the ASCII code, developed in the 1960s, was the de facto standard. Each character of text was assigned a 7-bit code (actually, a (7+1)-bit code, where the eighth bit serves as a parity, for added reliability). Thus, there are $2^7 = 128$ ASCII codes, for the letters, digits, some punctuation marks, and various control functions. In the 1970s and 1980s, inexpensive, high-resolution printers and display monitors came into being, where virtually any character can be displayed and printed. These developments were the motivation for the Unicode project which started in the early 1990s. Unicode assigns 16-bit codes to text symbols, and can therefore represent $2^{16} = 65{,}536$ symbols (there are provisions for even longer codes, so the number of possible Unicode symbols is much greater).

The point is that certain letters appear in text more often than others, and the use of fixed-size codes introduces structure (and thus redundancy) into text. It has been known for a long time that the letters `E`, `T`, and `A` are common in English texts, while

J, Z, and Q are rare. Thus, English text can be compressed by assigning variable-length codes to the various letters such that common letters are assigned short codes and rare letters are assigned long codes. Chapter 1 discusses a few variable-length codes and their applications.

Note that text compression must be lossless. It is hard to come up with examples where text data can lose a certain percentage of the text while being compressed, and still be useful after decompression. However, the other types of data discussed here can lose much data while being compressed, and be decompressed later without any noticeable degradation in quality. This is why lossy compression, which often features excellent performance, is so popular.

Images. A digital image is a rectangular array of dots called pixels. A pixel has one attribute, its color, and this attribute is stored in the computer as a fixed-size code. The use of fixed-size codes again introduces redundancy, because adjacent pixels tend to have similar codes (we say that the pixels are correlated). An image where adjacent pixels always have wildly different colors looks random, has no structure, features no redundancy, and therefore cannot be compressed. Images that are of interest, however, are far from random and exhibit structure in the form of pixel correlation. This type of redundancy is termed interpixel redundancy.

Thus, an image can be compressed by, for example, subtracting the values of adjacent pixels. The pixels have similar colors, so their differences are small numbers, which require fewer bits. More sophisticated approaches to image compression are discussed in Chapter 5.

In addition to interpixel redundancy, images often have two more types of redundancy, coding redundancy and psychovisual redundancy, which can be exploited for compression.

Coding redundancy has to do with the distribution of colors in an image. Given a digital image, it is easy to count the number of pixels that have color C. When this is done for every color C, we normally find that a few colors dominate the image. We say that the color distribution (or histogram) is nonuniform. This redundancy suggests a way to compress the image. Replace each pixel with a variable-length code and assign the short codes to the dominant colors.

Psychovisual redundancy has to do with the properties of the human eye. The eye is very sensitive to light and can often detect just a few photons. However, the eye is not a precision device and its sensitivity varies with the type of light that falls on it. It has been known for many years that the eye is very sensitive to variations in the luminance (brightness) of light but is much less sensitive to variations in the chrominance (color) component of the light. Thus, an image can be compressed if the color of each pixel is represented in terms of luminance and chrominance and the latter components are heavily quantized.

Video data consists of a string of images, much as a movie consists of many images (called frames) on a strip of celluloid. There are two sources of redundancy in a video, intraframe redundancy (the correlation of pixels in each frame) and interframe redundancy (the fact that adjacent frames tend to be similar). The former redundancy can be reduced by the same methods employed in image compression, while the latter redundancy is dealt with by methods that compare a frame with its predecessor and encode the differences between them.

Audio data also features redundancy in the form of correlation between consecutive audio samples, but first, a few words about audio and how it is digitized.

Sound is a wave. It can be viewed as a physical disturbance in the air (or some other media) or as a pressure wave propagated by the movement of molecules. A microphone is a device that senses sound and converts it to an electrical wave, i.e., a voltage that varies continuously with time in the same way as the sound. To convert this voltage into a format where it can be input into a computer, stored, edited, and played back, the voltage is sampled many times each second. Each sample is a number whose value is proportional to the voltage at the time of sampling. Figure Intro.1 shows a wave sampled at three points and it is obvious that the first sample is a small number and the third sample is a large number, close to the maximum.

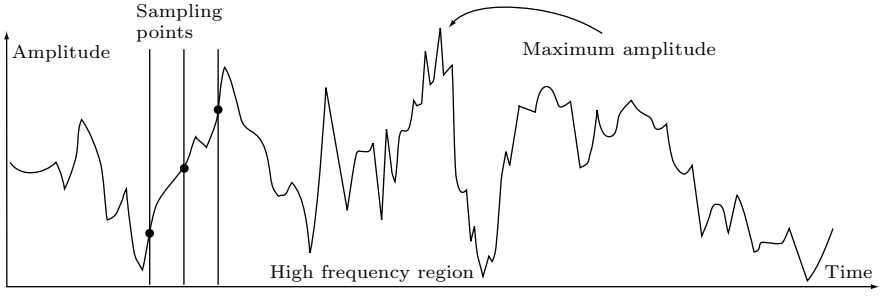

Figure Intro.1: Sound Wave and Three Samples.

Thus, audio sampling (or digitized sound) is a simple concept, but its success in practice depends on one important factor, the sampling rate. How many times should a sound wave be sampled each second? Sampling too often creates too many audio samples, while a very low sampling rate results in low-quality played-back sound. It seems intuitively that the sampling rate should depend on the frequency, but the frequency of a sound wave varies all the time, while the sampling rate should remain constant (a variable sampling rate makes it difficult to edit and play back the digitized sound). The solution was discovered back in the 1920s by H. Nyquist. It states that the optimum sampling frequency should be slightly greater than twice the maximum frequency of the sound. The sound wave of Figure Intro.1 has a region of high frequency at its center. To obtain the optimum sampling rate for this particular wave, we should determine the maximum frequency at this region, double it, and increase the result slightly.

Every sound wave has its own maximum frequency, but digitized sound used in practice is based on the fact that the highest frequency that the human ear can perceive is about 22,000 Hz. The optimum sampling rate that corresponds to this frequency is 44,100 Hz, and this rate is used when sound is digitized and recorded on a CD or DVD.

Now, back to audio compression. Digital audio is a string of audio samples, and it can be compressed because adjacent audio samples tend to be similar; they are correlated, which introduces redundancy into the audio data. With 44,100 samples each second, it is no wonder that adjacent samples are virtually always similar. Audio data where many audio samples are very different from their neighbors would sound harsh and dissonant.

Thus, audio can be compressed by subtracting each audio sample from its predecessor and replacing the differences (which tend to be small integers) by suitable variable-length codes. Practical methods often "predict" the current sample by computing a weighted sum of several previously-input samples, and then subtracting the current sample from the prediction.

Entropy and Redundancy

Understanding data compression and its codes must start with an understanding of information, because the former is based on the latter. This short section introduces the relevant concepts from information theory.

Information theory is the creation, in the late 1940s, of Claude Shannon. Shannon tried to develop means for measuring the amount of information stored in a symbol without considering the meaning of the information. He discovered the connection between the logarithm function and information, and showed that the information content (in bits) of a symbol with probability p is $-\log_2 p$. If the base of the logarithm is e, then the information is measured in units called nats. If the base is 3, the information units are trits, and if the base is 10, the units are referred to as Hartleys.

Information theory is concerned with the transmission of information from a sender (termed a source), through a communications channel, to a receiver. The sender and receiver can be persons or machines and the receiver may, in turn, act as intermediary and send the information it has received to another receiver. The information is sent in units called symbols (normally bits, but in verbal communications the symbols are spoken words) and the set of all possible data symbols is an alphabet.

The most important single factor affecting communications is noise in the communications channel. In verbal communications, this noise is, literally, noise. When trying to talk in a noisy environment, we may lose part of the discussion. In electronic communications, the channel noise is caused by imperfect hardware and by factors such as sudden lightning, voltage fluctuations—old, high-resistance wires—sudden surge in temperature, and interference from machines that generate strong electromagnetic fields.

The presence of noise implies that special codes should be used to increase the reliability of transmitted information. This is referred to as channel coding or, in everyday language, error-control codes.

The second most important factor affecting communications is sheer volume. Any communications channel has a limited capacity. It can transmit only a limited number of symbols per time unit. An obvious way to increase the amount of data transmitted is to compress it before it is sent (in the source). Methods to compress data are therefore known as source coding or, in everyday language, data compression. The feature that makes it possible to compress data is the fact that individual symbols appear in our data files with different probabilities. Thus, data can be compressed by assigning variable-length codes to the individual data symbols such that short codes are assigned to the common symbols.

Two concepts from information theory, namely entropy and redundancy, are needed in order to fully understand the principles behind the various methods for and approaches to data compression.

Roughly speaking, the term "entropy" as defined by Shannon is a real number that is proportional to the minimum number of yes/no questions needed to reach the answer to some question. Another way of looking at entropy is as a quantity that describes how much information is included in a signal or an event.

In order to understand the definition of entropy, we perform a thought experiment where we measure the heights of 10,000 people. Suppose that we find that 1,500 people have a height of h. We can say that the probability of having height h in our sample of 10,000 people is $1{,}500/10{,}000 = 0.15$. Statisticians perform such experiments and they talk about random variables. A random variable X is an entity that can have certain values x_i, each with probability P_i. In our experiment, the probability that our random variable will have the value h is 0.15, and it can have other values with different probabilities.

Given a discrete random variable X that can have n values x_i with probabilities P_i, the entropy $H(X)$ of X is defined as

$$H(X) = -\sum_{i=1}^{n} P_i \log_2 P_i.$$

The surprising, unexpected part in this definition is the use of the logarithm. The following paragraphs explain why the familiar logarithm function constitutes such an important part of information theory and plays such an important role in measuring information.

Imagine a source that emits symbols a_i with probabilities p_i. We assume that the source is memoryless, i.e., the probability of a symbol being emitted does not depend on what has been emitted in the past (the parallel in our thought experiment is that the height of a person being measured does not depend on the height of the previous person measured). We want to define a function $I(a_i)$ that will measure the amount of information gained when we discover that the source has emitted symbol a_i. Function I will also measure our uncertainty as to whether the next symbol will be a_i. Alternatively, $I(a_i)$ corresponds to our surprise in finding that the next symbol is a_i. Clearly, our surprise at seeing a_i emitted is inversely proportional to the probability p_i (we are surprised when a low-probability symbol is emitted, but not when we notice a high-probability symbol). Thus, it makes sense to require that function I satisfies the following conditions:

1. $I(a_i)$ is a decreasing function of p_i, and returns 0 when the probability of a symbol is 1. This reflects our feeling that high-probability events convey less information.

2. $I(a_i a_j) = I(a_i) + I(a_j)$. This is a result of the source being memoryless and the probabilities being independent. Discovering that a_i was immediately followed by a_j, provided us with the same information as knowing that a_i and a_j were emitted independently.

Even those with a minimal mathematical background may quickly realize that the logarithm function satisfies the two conditions above. This is the first example of the relation between the logarithm function and the quantitative measure of information. The next few paragraphs illustrate other connections between the two.

Consider the case of person A selecting at random an integer N between 1 and 64 and person B having to guess N. What is the minimum number of yes/no questions that are needed for B to guess N? Those familiar with the technique of binary search know the answer. Using this technique, B should divide the interval 1–64 in two, and should start by asking "is N between 1 and 32?" If the answer is no, then N is in the interval 33 to 64. This interval is then divided by two and B's next question should be "is N between 33 and 48?" This process continues until the interval selected by B shrinks to a single number.

It does not take much to see that exactly six questions are necessary to determine N. This is because 6 is the number of times 64 can be divided in half. Mathematically, this is equivalent to writing $2^6 = 64$ or $6 = \log_2 64$, which is why the logarithm is the mathematical function that quantifies information.

> What we call reality arises in the last analysis from the posing of yes/no questions. All things physical are information-theoretic in origin, and this is a participatory universe.
> —John Wheeler

Another approach to the same problem is to consider a nonnegative integer N and ask how many digits does it take to express it. The answer, of course, depends on N. The greater N, the more digits are needed. The first 100 nonnegative integers (0 through 99) can be expressed by two decimal digits. The first 1,000 such integers can be expressed by three digits. Again it does not take long to see the connection. The number of digits required to represent N equals approximately $\log N$. The base of the logarithm is the same as the base of the digits. For decimal digits, use base 10; for binary digits (bits), use base 2. If we agree that the number of digits it takes to express N is proportional to the information content of N, then again the logarithm is the function that gives us a measure of the information. As an aside, the precise length, in bits, of the binary representation of a positive integer n is $1+\lfloor\log_2 n\rfloor$, or alternatively, $\lceil\log_2(n+1)\rceil$. When n is represented in any other number base b, its length is given by the same formula, but with the logarithm in base b instead of 2.

Here is another observation that illuminates the relation between the logarithm and information. A 10-bit string can have $2^{10} = 1,024$ values. We say that such a string may contain one of 1,024 messages, or that the length of the string is the logarithm of the number of possible messages the string can convey.

The following example sheds more light on the concept of entropy and will prepare us for the definition of redundancy. Given a set of two symbols a_1 and a_2, with probabilities P_1 and P_2, respectively, we compute the entropy of the set for various values of the probabilities. Since $P_1+P_2 = 1$, the entropy of the set is $-P_1\log_2 P_1-(1-P_1)\log_2(1-P_1)$ and the results are summarized in Table Intro.2.

When $P_1 = P_2$, at least one bit is required to encode each symbol, reflecting the fact that the entropy is at its maximum, the redundancy is zero, and the data cannot be compressed. However, when the probabilities are very different, the minimum number of bits required per symbol drops significantly. We may not be able to conceive of a compression method that expresses each symbol in just 0.08 bits, but we know that when $P_1 = 99\%$, such compression is theoretically possible.

In general, the entropy of a set of n symbols depends on the individual probabilities P_i of the symbols and is largest when all n probabilities are equal. Data representations

P_1	P_2	Entropy
0.99	0.01	0.08
0.90	0.10	0.47
0.80	0.20	0.72
0.70	0.30	0.88
0.60	0.40	0.97
0.50	0.50	1.00

Table Intro.2: Probabilities and Entropies of Two Symbols.

often include redundancies and data can be compressed by reducing or eliminating these redundancies. When the entropy is at its maximum, the data has maximum information content and therefore cannot be further compressed. Thus, it makes sense to define redundancy as a quantity that goes down to zero as the entropy reaches its maximum.

> The fundamental problem of communication is that of reproducing at one point either exactly or approximately a message selected at another point.
>
> —Claude Shannon (1948)

To understand the definition of redundancy, we start with an alphabet of symbols a_i, where each symbol appears in the data with probability P_i. The data is compressed by replacing each symbol with an l_i-bit-long code. The average code length is the sum $\sum P_i l_i$ and the entropy (the smallest number of bits required to represent the symbols) is $\sum[-P_i \log_2 P_i]$. The redundancy R of the set of symbols is defined as the average code length minus the entropy. Thus,

$$R = \sum_i P_i l_i - \sum_i [-P_i \log_2 P_i].$$

The redundancy is zero when the average code length equals the entropy, i.e., when the codes are the shortest and compression has reached its maximum.

Given a set of symbols (an alphabet), we can assign binary codes to the individual symbols. It is easy to assign long codes to symbols, but most practical applications require the shortest possible codes.

Consider the four symbols a_1, a_2, a_3, and a_4. If they appear in our data strings with equal probabilities ($= 0.25$), then the entropy of the data is $-4(0.25 \log_2 0.25) = 2$. Two is the smallest number of bits needed, on average, to represent each symbol in this case. We can simply assign our symbols the four 2-bit codes 00, 01, 10, and 11. Since the probabilities are equal, the redundancy is zero and the data cannot be compressed below two bits/symbol.

Next, consider the case where the four symbols occur with different probabilities as shown in Table Intro.3, where a_1 appears in the data (on average) about half the time, a_2 and a_3 have equal probabilities, and a_4 is rare. In this case, the data has entropy $-(0.49 \log_2 0.49 + 0.25 \log_2 0.25 + 0.25 \log_2 0.25 + 0.01 \log_2 0.01) \approx -(-0.050 - 0.5 - 0.5 - 0.066) = 1.57$. The smallest number of bits needed, on average, to represent each symbol has dropped to 1.57.

If we again assign our symbols the four 2-bit codes 00, 01, 10, and 11, the redundancy would be $R = -1.57 + \log_2 4 = 0.43$. This suggests assigning variable-length codes to

Symbol	Prob.	Code1	Code2
a_1	0.49	1	1
a_2	0.25	01	01
a_3	0.25	010	000
a_4	0.01	001	001

Table Intro.3: Variable-Length Codes.

the symbols. Code1 of Table Intro.3 is designed such that the most common symbol, a_1, is assigned the shortest code. When long data strings are transmitted using Code1, the average size (the number of bits per symbol) is $1 \times 0.49 + 2 \times 0.25 + 3 \times 0.25 + 3 \times 0.01 = 1.77$, which is very close to the minimum. The redundancy in this case is $R = 1.77 - 1.57 = 0.2$ bits per symbol. An interesting example is the 20-symbol string $a_1a_3a_2a_1a_3a_3a_4a_2a_1a_1a_2a_2a_1a_1a_3a_1a_1a_2a_3a_1$, where the four symbols occur with approximately the right frequencies. Encoding this string with Code1 yields the 37 bits:

1|010|01|1|010|010|001|01|1|1|01|01|1|1|010|1|1|01|010|1

(without the vertical bars). Using 37 bits to encode 20 symbols yields an average size of 1.85 bits/symbol, not far from the calculated average size. (The reader should bear in mind that our examples are short. To obtain results close to the best that's theoretically possible, an input stream with at least thousands of symbols is needed.)

However, the conscientious reader may have noticed that Code1 is bad because it is not a prefix code. Code2, in contrast, is a prefix code and can be decoded uniquely. Notice how Code2 was constructed. Once the single bit 1 was assigned as the code of a_1, no other codes could start with 1 (they all had to start with 0). Once 01 was assigned as the code of a_2, no other codes could start with 01. This is why the codes of a_3 and a_4 had to start with 00. Naturally, they became 000 and 001.

Several important data compression terms are introduced next.

- The *compressor* or *encoder* is the program that compresses raw data and generates compressed (low-redundancy) output. The *decompressor* or *decoder* converts in the opposite direction. Note that the term *encoding* is very general and has several meanings, but since this book discusses only data compression, it employs the term *encoder* for compressor. The term *codec* is used to describe both the encoder and the decoder. Similarly, the term *companding* is short for "compressing/expanding."

- For the original, uncompressed data, we use the terms *unencoded*, *raw*, or *original* data. The compressed data is also termed *encoded*. The term *bitstream* is often used in the literature to indicate the compressed data.

- A *nonadaptive* compression method is inflexible and does not modify its operations, its parameters, or its tables in response to the particular data being compressed. In contrast, an *adaptive* method examines the raw data and modifies its operations and/or its parameters accordingly. Some compression methods use a 2-pass algorithm, where the first pass reads the input to collect statistics on the data to be compressed, and the

second pass does the actual compression using parameters or codes set by the first pass. Such a method may be called *semiadaptive*. A data compression method can also be *locally adaptive*, meaning it adapts itself to local conditions in the input and varies this adaptation as it moves from region to region in the input.

- *Lossy/lossless compression:* Certain compression methods are lossy. They achieve better compression by losing some information. When the compressed data is later decompressed, the result is different from the original. Such a method makes sense especially in image, video, or audio compression. If the loss of data is small, the eye or ear may not perceive any difference. In contrast, text files, especially files containing computer programs, often become meaningless or worthless if even one bit is modified. Such files should be compressed only by a lossless compression method.

- *Perceptive compression:* A lossy encoder must take advantage of the special type of data that is being compressed. It should delete only data whose absence would not be detected by our senses. Such an encoder must therefore employ algorithms based on our understanding of psychoacoustic and psychovisual perception, which is why it is sometimes referred to as a perceptive encoder. Such an encoder can be made to operate at a constant compression ratio, where for each x bits of raw data, it outputs y bits of compressed data. This is convenient in cases where the compressed data has to be transmitted at a constant rate. The trade-off is a variable subjective quality. Parts of the original data that are difficult to compress may, after decompression, look (or sound) bad. Such parts may require more than y bits of output for x bits of input.

- *Symmetric compression* is the case where the decompressor is the reverse of the compressor. Such a method makes sense for general work, where the same number of files is compressed as is decompressed. In an asymmetric compression method, either the compressor or the decompressor may have to work significantly harder. Such methods have their uses and are not necessarily bad. A compression method where the compressor executes a slow, complex algorithm and the decompressor is simple is a natural choice when files are compressed into an archive (a CDs and DVDs are good examples) where they will be decompressed and used very often. The opposite case is useful in environments where files are updated all the time and backups are made. There is only a small chance that a backup file will be used, so the decompressor is rarely used and can be slow.

When you look	kool uoy nehW
into a mirror	rorrim a otni
it is not	ton si ti
yourself you see	ees uoy flesruoy
but a kind	dnik a tub
of apish error	rorre hsipa fo
posed in fearful	lufraef ni desop
symmetry	yrtemmys

John Updike, "Mirror," in *Telephone Poles and Other Poems* (1963)

◇ **Exercise Intro.1:** Give an example of a compressed file where efficient compression is important but the speed of both compressor and decompressor isn't important.

■ Many modern compression methods are asymmetric. Often, the formal specification of such a method consists of a description of the decoder and the format of the compressed data, but does not discuss the operation of the encoder. Any encoder that generates a correct compressed file is considered compliant, as is also any decoder that can read and decode such a file. The advantage of such a description is that anyone is free to develop and implement new, sophisticated algorithms for the encoder. The implementor need not even publish the details of the encoder and may consider it proprietary. If a compliant encoder is demonstrably better than competing encoders, it may become a commercial success. In such a scheme, the encoder is considered *algorithmic*, while the decoder, which is normally much simpler, is termed *deterministic*.

■ A data compression method is called *universal* if the compressor and decompressor do not know the statistics of the input data and do not use it explicitly. A universal method is *optimal* if the compressor can produce compression factors that asymptotically approach the entropy of the input stream for long inputs.

■ The term *file differencing* refers to any method that locates and compresses the differences between two files. Imagine a file A with two copies that are kept by two users. When a copy is updated by one user, it should be sent to the other user, to keep the two copies identical. Instead of sending a copy of A, which may be big, a much smaller file containing just the differences, in compressed format, can be sent and used at the receiving end to update the copy of A.

■ Most compression methods operate in the *streaming mode*, where the codec inputs a byte or several bytes, processes them, and continues until an end-of-file is sensed. Some methods, such as Burrows–Wheeler (Section 7.1), work in the *block mode*, where the input is read block by block and each block is encoded separately. The block size in this case should be a user-controlled parameter, because its size may greatly affect the performance of the method.

■ *Compression performance*: Several measures are commonly used to indicate the performance of a compression method.

1. The *compression ratio* is defined as

$$\text{Compression ratio} = \frac{\text{size of the output stream}}{\text{size of the input stream}}.$$

A value of 0.6 means that the data occupies 60% of its original size after compression. Values greater than 1 imply expansion (negative compression). The compression ratio can also be called bpb (bit per bit), since it equals the number of bits in the compressed data that are needed, on average, to compress one bit in the input data. In modern, efficient text compression methods, it makes sense to talk about bpc (bits per character)—the number of bits it takes, on average, to compress one character in the input.

The term *bitrate* is a general name for bpb and bpc. Thus, the main goal of data compression is to represent any given data at low bitrates.

2. The inverse of the compression ratio is the *compression factor*:

$$\text{Compression factor} = \frac{\text{size of the input stream}}{\text{size of the output stream}}.$$

In this case, values greater than 1 indicate compression and values less than 1 imply expansion. This measure seems natural to many people, since the bigger the factor, the better the compression.
3. The expression $100 \times (1 - \text{compression ratio})$ is also a reasonable measure of compression performance. A value of 60 means that the output occupies 40% of its original size (or that the compression has resulted in savings of 60%).
4. In image compression, the quantity bpp (bits per pixel) is commonly used. It equals the number of bits needed, on average, to compress one pixel of the image. This quantity should always be compared with the bpp before compression.
5. The *compression gain* is defined as

$$100 \log_e \frac{\text{reference size}}{\text{compressed size}},$$

where the reference size is either the size of the input or the size of the compressed data produced by some standard lossless compression method. For small numbers x, it is true that $\log_e(1 + x) \approx x$, so a small change in a small compression gain is very similar to the same change in the compression ratio. Because of the use of the logarithm, two compression gains can be compared simply by subtracting them. The unit of the compression gain is called *percent log ratio* and is denoted by $\frac{\circ}{\circ}$.
6. The speed of compression can be measured in *cycles per byte* (CPB). This is the average number of machine cycles it takes to compress one byte. This measure is important when compression is done by special hardware.
7. Other quantities, such as mean square error (MSE) and peak signal-to-noise ratio (PSNR), are used to measure the distortion caused by lossy compression of still images and video.

■ The *probability model*. This concept is important in statistical data compression methods. In such a method, a model for the data has to be constructed before compression can begin. A typical model may be built by reading the entire input stream, counting the number of times each symbol appears (its frequency of occurrence), and computing the probability of occurrence of each symbol. The data is then input again, symbol by symbol, and is compressed using the information in the probability model.

Reading the entire input twice is slow, which is why practical compression methods use estimates, or adapt themselves to the data as it is being input and compressed. It is easy to scan large quantities of, say, English text and compute the frequencies and probabilities of every character. This information can later serve as an approximate model for English text and can be used by text compression methods to compress any English text. It is also possible to start by assigning equal probabilities to all the symbols in an alphabet, then reading symbols and compressing them, and, while doing that, also counting frequencies and changing the model as compression progresses. This is the principle behind adaptive compression methods.

- The term "baud" is used in this book to mean bits per second, but see a more general definition in `http://en.wikipedia.org/wiki/Baud`.

Data Compression Resources

A vast number of resources on data compression is available. Any Internet search under "data compression," "lossless data compression," "image compression," "audio compression," and similar topics returns at least tens of thousands of results. The following URLs have useful links and pointers to the many data compression resources available on the Internet and elsewhere:

`http://www.hn.is.uec.ac.jp/~arimura/compression_links.html`
`http://cise.edu.mie-u.ac.jp/~okumura/compression.html`
`http://compression.ca/` (mostly comparisons), and `http://datacompression.info/`

The latter URL has a wealth of information on data compression, including tutorials, links to workers in the field, and lists of books. The site was maintained by Mark Nelson but it currently belongs to Visicron Corp.

Traditional (hardcopy) resources range from general texts and texts on specific aspects or particular methods, to survey articles in magazines, to technical reports and research papers in scientific journals. Following is a short list of (mostly general) books, sorted by date of publication.

James Storer, *Proceedings of the IEEE Data Compression Conference*, IEEE Press, published annually since 1991.

Tinku Acharya and Ping-Sing Tsai, *JPEG2000 Standard for Image Compression: Concepts, Algorithms and VLSI Architectures*, John Wiley and Sons (2005).

Ida Mengyi Pu, *Fundamental Data Compression*, Butterworth-Heinemann (2005).

Khalid Sayood, *Introduction to Data Compression*, Morgan Kaufmann, 3rd edition (2005).

Darrel Hankerson, *Introduction to Information Theory and Data Compression*, Chapman & Hall (CRC), 2nd edition (2003).

Peter Symes, *Digital Video Compression*, McGraw-Hill/TAB Electronics (2003).

Charles Poynton, *Digital Video and HDTV Algorithms and Interfaces*, Morgan Kaufmann (2003).

Iain E. G. Richardson, *H.264 and MPEG-4 Video Compression: Video Coding for Next Generation Multimedia*, John Wiley and Sons (2003).

Marina Bosi and Richard E. Goldberg, *Introduction to Digital Audio Coding and Standards*, Springer Verlag (2003).

Khalid Sayood, *Lossless Compression Handbook*, Academic Press (2002).

Touradj Ebrahimi and Fernando Pereira, *The MPEG-4 Book*, Prentice Hall (2002).

Adam Drozdek, *Elements of Data Compression*, Course Technology (2001).

Alistair Moffat and Andrew Turpin, *Compression and Coding Algorithms*, Springer Verlag (2002).

David Taubman and Michael Marcellin (Editors), *JPEG2000: Image Compression Fundamentals, Standards and Practice*, Springer Verlag (2001).

Kamisetty R. Rao, *The Transform and Data Compression Handbook*, CRC (2000).

Ian H. Witten, Alistair Moffat, and Timothy C. Bell, *Managing Gigabytes: Compressing and Indexing Documents and Images*, Morgan Kaufmann, 2nd edition (1999).

Peter Wayner, *Compression Algorithms for Real Programmers*, Morgan Kaufmann (1999).

John Miano, *Compressed Image File Formats: JPEG, PNG, GIF, XBM, BMP*, ACM Press and Addison-Wesley Professional (1999).

Jerry D. Gibson et al. *Digital Compression for Multimedia: Principles & Standards*, Morgan Kaufmann (1998).

Nikil Jayant, *Signal Compression: Coding of Speech, Audio, Text, Image and Video*, World Scientific (1997).

Weidong Kou, *Digital Image Compression: Algorithms and Standards*, Kluwer (1995).

Mark Nelson and Jean-Loup Gailly, *The Data Compression Book*, M&T Books, 2nd edition (1995).

Rafail Krichevsky, *Universal Compression and Retrieval*, Kluwer Academic Publishers, 1994.

William B. Pennebaker and Joan L. Mitchell, *JPEG: Still Image Data Compression Standard*, Springer Verlag (1992).

Timothy C. Bell, John G. Cleary, and Ian H. Witten, *Text Compression*, Prentice Hall (1990).

James A. Storer, *Data Compression: Methods and Theory*, Computer Science Press (1988).

John Woods, ed., *Subband Coding*, Kluwer Academic Press (1990).

The symbol "␣" is used to indicate a blank space in places where spaces may lead to ambiguity.

Comments, suggestions, and corrections are always welcome and should be directed to `dsalomon@csun.edu`.

History is a kind of introduction to more interesting people than we can possibly meet in our restricted lives; let us not neglect the opportunity.

—Dexter Perkins