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

Simple Repetition Suppression

If in a sequence a series on *n* successive tokens appears we can replace these with a token and a count number of occurrences. We usually need to have a special ***flag*** to denote when the repeated token appears

For Example

89400000000000000000000000000000

we can replace with

894f32

where f is the flag for zero.

Compression savings depend on the content of the data.

Applications of this simple compression technique include:

- Suppression of zero's in a file (***Zero Length Supression***)
  - Silence in audio data, Pauses in conversation ***etc.***
  - Bitmaps
  - Blanks in text or program source files
  - Backgrounds in images
- other regular image or data tokens

VARIABLE-SIZE CODES

Consider the four symbols **a1, a2, a3, and a4.** 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 2 bits/symbol.

Next, consider the case where the four symbols occur with **different probabilities** as shown in Table 2.2. In this case, the data has entropy

$-(0.49\ \log2\ 0.49+0.25\ \log2\ 0.25+0.25\ \log2\ 0.25+0.01\ \log2\ 0.01) \approx -(-0.050-0.5-0.5-0.066) = 1.57$

The smallest number of bits needed, on average, to represent each symbol has dropped to 1.57.

Symbol	Prob.	Code1	Code2
a_1	.49	1	1
a_2	.25	01	01
a_3	.25	010	000
a_4	.01	001	001

Table 2.2: Variable-Size Codes.

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-size codes to the symbols. Code1 of Table 2.2 is designed such that the most common symbol, **a1, 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

Redundancy = Average – Entropy

$$R = 1.77 - 1.57 = 0.2 \text{ bits per symbol.}$$

An interesting example is the 20-symbol string

a1a3a2a1a3a3a4a2a1a1a2a2a1a1a3a1a1a2a3a1,

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 by using

table 2-3, an average size of

.45*1+.25*2+.25*3+.05*3=1.85 bits/symbol,

not far from the calculated average size.

Table 2-3

Symbol	Prob.	Code1	Code2
a1	.45	1	1
a2	.25	01	01
a3	.25	010	000
a4	.05	001	001

However, when we try to decode the binary string above, it becomes obvious that Code1 is bad. The first bit is 1, and since only a1 is assigned this code, it (a1) must be the first symbol. The next bit is 0, but the codes of a2, a3, and a4 all start with a 0, so the decoder has to read the next bit. It is 1, but the codes of both a2 and a3 start with 01. The decoder does not know whether to decode the string as 1|010|01 . . ., which is a1a3 a2 . . ., or as 1|01|001 . . ., which is a1a2a4 . . .. Code1 is thus ambiguous. In contrast, Code2, which has the same average size as Code1, can be decoded unambiguously. This property requires that once a certain bit pattern has been assigned as the code of a symbol, no other codes should start with that pattern

(the pattern cannot be the prefix of any other code). Once the string "1" was assigned as the code of a1, no other codes could start with 1 (i.e., they all had to start with 0). Once "01" was assigned as the code of a2, no other codes could start with 01. This is why the codes of a3 and a4 had to start with 00. Naturally, they became 000 and 001.

Following these principles produces short, unambiguous codes, but not necessarily the best (i.e., shortest) ones. In addition to these principles, an algorithm is needed that always produces a set of shortest codes (ones with the minimum average size). The only input to such an algorithm is the frequencies (or the probabilities) of the symbols of the alphabet.
Two such algorithms, the Shannon-Fano method and the Huffman method (It should be noted that not all statistical compression methods assign variable size codes to the individual symbols of the alphabet. A notable exception is arithmetic coding.)

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.
We can define the ***entropy*** of a single symbol

ai as $-$***Pi log2 Pi***.

This is the ***smallest number of bits needed***, on average, to represent the symbol. In information theory, ***entropy*** measures the ***amount of uncertainty of an unknown or random quantity***.
The entropy of data depends on the individual probabilities Pi and is largest when all n probabilities are equal. This fact is used to define the ***redundancy R*** in the data. It is defined as the ***difference*** between a ***symbol set's largest possible entropy and its actual entropy***. Thus

$$R = \left[-\sum_{1}^{n} P \log_2 P\right] - \left[-\sum_{1}^{n} P_i \log_2 P_i\right] = \log_2 n + \sum_{1}^{n} P_i \log_2 P_i.$$

Thus, the test for ***fully compressed data (no redundancy)*** is:

$$\log_2 n + \sum_1^n P_i \log_2 P_i = 0$$

However, when we try to ***decode*** the binary string above, it becomes obvious that ***Code1*** is ***bad***. WHY?

The first bit is ***1***, and since only a_1 is assigned this code, (a_1) must be the first symbol.

- The next bit is ***0***, but the codes of ***a2, a3, and a4*** all start with a ***0***, so the ***decoder*** has to ***read the next bit***. It is ***1***, but the codes of both ***a2 and a3*** start with ***01***. The ***decoder does not know whether to decode the string as***
1|010|01 . . ., which is ***a1a3 a2*** . . .,
or as

1|01|001 . . ., which is ***a1a2a4*** . . ..

In contrast, ***Code2***, which has ***the same average size as Code1***, can be ***decoded unambiguously***. This ***property requires*** that ***once*** a certain ***bit pattern*** has been ***assigned*** as the code of a symbol, ***no other codes*** should ***start*** with ***that pattern (the pattern cannot be the prefix of any other code).***
- Once the string "***1***" was assigned as the code of ***a1***, ***no other codes could start with 1*** (i.e., ***they all had to start with 0***).

- Once "***01***" was assigned as the code of ***a2***, ***no other codes*** could start with ***01***. This is why the codes of ***a3 and a4 had to start with 00***. Naturally, they became ***000*** and ***001***.

Designing ***variable-size codes*** is therefore done by following two principles:

(1) Assign ***short codes*** to the ***more frequent symbols*** and
(2) Obey the ***prefix property***.

Following these principles produces ***short***, ***unambiguous*** codes, but not ***necessarily the best*** (i.e., shortest) ones. In addition to these principles, an algorithm is needed that always produces a set of shortest codes (ones with the minimum average size). The only input to such an algorithm is the frequencies (or the probabilities) of the symbols of the alphabet.

THE UNARY CODE

The ***unary*** code of the positive ***integer n*** is defined as ***n − 1 ones*** followed by a ***single 0*** (Table 2.3) or, alternatively, as n − 1 zeros followed by a single one. The length of the unary code for the integer n is therefore n bits.

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

Table 2.3: Some Unary Codes.

It is also possible to define general unary codes, also known as ***start-step-stop codes***. Such a code depends on a triplet (start, step, stop) of integer parameters and is defined as follows:

1. Codewords are created to code symbols used in the data, such that the ***nth codeword*** consists of ***n ones***,
2. followed by ***one 0***,
3. Followed by all the combinations of ***a bits*** where ***a = start + n × step***. If ***a = stop***, then the ***single 0 preceding the a bits is dropped***. The number of codes for a given triplet is finite and depends on the choice of parameters. Tables 2.4 and 2.5 show the ***680*** codes of (***3,2,9***) and the ***2044*** codes of (***2,1,10***).

n	$a = 3 + n \cdot 2$	nth codeword	Number of codewords	Range of integers
0	3	$0xxx$	$2^3 = 8$	0–7
1	5	$10xxxx$	$2^5 = 32$	8–39
2	7	$110xxxxxx$	$2^7 = 128$	40–167
3	9	$111xxxxxxxx$	$2^9 = 512$	168–679
		Total	680	

Table 2.4: The General Unary Code (3,2,9).