Appendix C

Data Compression Using ZIP

ZIP is a simple matching algorithm using two sliding windows, called the *base window* and the *look-ahead window*. These two windows are placed side-by-side on the data file, where the look-ahead window goes ahead of the base window. ZIP scans the entire file by sliding these two windows and encoding data on the fly. In particular, ZIP finds the longest prefix s of the data string contained in the look-ahead window that also appears in the base window. This string in the look-ahead window (if found) is a copy of s in the base window, and so it can be uniquely identified by two attributes: (1) the distance between the location of the first character of s in the base window and the location of the first character in the look-ahead window and (2) the length of s. If the space needed to hold the values of these two attributes is smaller than the space needed to hold s, we obtain a saving of space.

To implement this idea, we will need to distinguish the binary values of the two attributes from normal encodings of characters. Suppose that the data file is encoded using the 8-bit ASCII code set. If the first bit is used as a parity bit, then it could be either 0 or 1. The first bit of the binary string representing the two attributes can also be either 0 or 1. Thus, to make a distinction, we add an extra bit of 1 in front of each ASCII code to yield a 9-bit extended ASCII code and add an extra bit of 0 in front of the binary string representing the two attributes. This simple encoding uniquely identifies the original data file.

In particular, let w_1 denote the number of characters the base window can hold, where $2^{d-1} < w_1 \leq 2^d$ for some $d \geq 1$. Let w_2 denote the number of characters the look-ahead window can hold, where $2^{l-1} < w_2 \leq 2^l$ for some l with $1 \leq l \leq d$. This produces a $(d + l + 1)$-bit binary encoding for s, where the first bit is 0 (used as an indicator), the next d bits represent the distance, and the last l bits represent the length. For convenience, we call this $(d + l + 1)$-bit code a *location code*.

A location code is easily distinguishable from any 9-bit extended ASCII code because a location code has a fixed length and an indicator 0 different from the indicator in a 9-bit extended ASCII code. In other words, given a compressed file using this encoding method, it can be uniquely and easily "uncompressed" back to its original ASCII format. The proof is left to the reader (see the Exercise). Thus, as long as $d + l + 1 < 8k$, where $k = |s|$, ZIP may save space. ZIP then shifts both of the base window and the look-ahead window to the right

Introduction to Network Security: Theory and Practice, Second Edition. Jie Wang and Zachary A. Kissel.

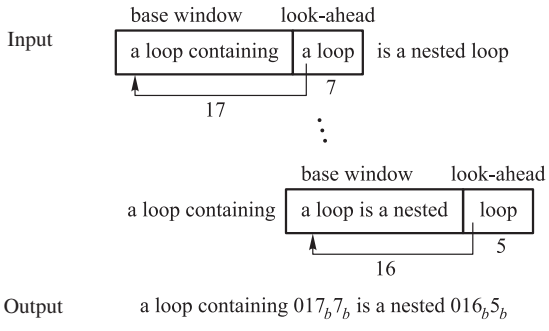

Figure C.1 A demonstration of a ZIP process

$\max\{1, k\}$ times and repeats the same procedure until the look-ahead window is shifted out of the data file.

For example, let $w_1 = 18$ and $w_2 = 7$. Then $d = 5$ and $l = 3$. Let us consider the following text string:

$$\text{“a loop containing a loop is a nested loop”}$$

Denote by n_b, the binary representation of positive integer n. Running ZIP on this character string produces the following output (see Figure C.1):

$$\text{“a loop containing } 017_b7_b \text{ is a nested } 016_b5_b\text{”},$$

where each letter and space in the output string is encoded by a 9-bit extended ASCII code. For clarity, we do not spell out the location code in binary. The length of the “compressed” string in binary is therefore equal to $18 \times 9 + 9 + 11 \times 9 + 9 = 279$ bits, and the length of the original character string encoded in the 8-bit ASCII code set is equal to $41 \times 8 = 328$ bits. Thus, ZIP has compressed the original data string to a shorter binary string.

To decode a compressed file, ZIP scans it from the beginning, removes the leading 1 from each 9-bit extended code, and replaces each 9-bit code with leading 0 by the corresponding character substring using the distance and length attributes.

Exercise

Show why the 10-bit code defined in this appendix is easily distinguishable from the extended 9-bit ASCII code. That is, given a compressed file using this encoding method, show that it can be uniquely and easily “uncompressed” back to its original ASCII format.