American Journal of Engineering Research (AJER)
e-ISSN : 2320-0847 p-ISSN : 2320-0936
Volume-03, Issue-02, pp-22-26
www.ajer.org

Research Paper Open Access

LZW Data Compression

Dheemanth H N,
Dept of Computer Science, National Institute of Engineering, Karnataka, India

Abstract: - Lempel–Ziv–Welch (LZW) is a universal lossless data compression algorithm created by Abraham Lempel, Jacob Ziv, and Terry Welch. LZW compression is one of the Adaptive Dictionary techniques. The dictionary is created while the data are being encoded. So encoding can be done on the fly. The dictionary need not be transmitted. Dictionary can be built up at receiving end on the fly. If the dictionary overflows then we have to reinitialize the dictionary and add a bit to each one of the code words. Choosing a large dictionary size avoids overflow, but spoils compressions. A codebook or dictionary containing the source symbols is constructed. For 8-bit monochrome images, the first 256 words of the dictionary are assigned to the gray levels 0-255. Remaining part of the dictionary is filled with sequences of the gray levels.LZW compression works best when applied on monochrome images and text files that contain repetitive text/patterns.

Keywords: - Encoding,Decoding,Compression Ratio

I. ENCODING

A dictionary is initialized to contain the single-character strings corresponding to all the possible input characters (and nothing else except the clear and stop codes if they're being used). The algorithm works by scanning through the input string for successively longer substrings until it finds one that is not in the dictionary. When such a string is found, the index for the string less the last character (i.e., the longest substring that is in the dictionary) is retrieved from the dictionary and sent to output, and the new string (including the last character) is added to the dictionary with the next available code. The last input character is then used as the next starting point to scan for substrings.

In this way, successively longer strings are registered in the dictionary and made available for subsequent encoding as single output values. The algorithm works best on data with repeated patterns, so the initial parts of a message will see little compression. As the message grows, however, the compression ratio tends asymptotically to the maximum.

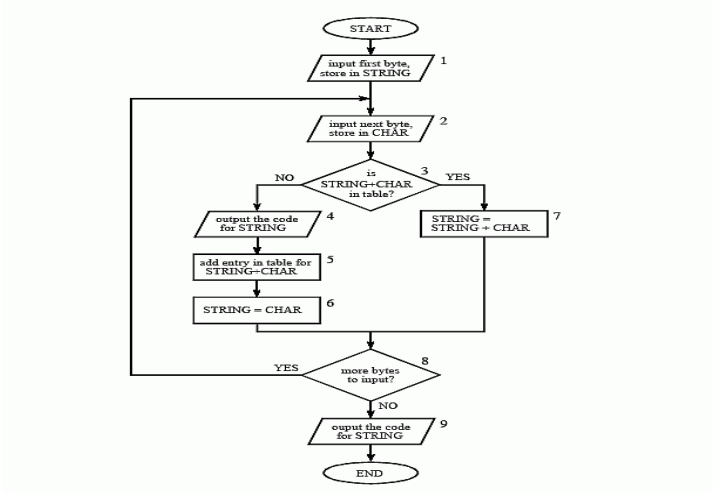


FIGURE 27-7
LZW compression flowchart. The variable, *CHAR*, is a single byte. The variable, *STRING*, is a variable length sequence of bytes. Data are read from the input file (box 1 & 2) as single bytes, and written to the compressed file (box 4) as 12 bit codes. Table 27-3 shows an example of this algorithm.

II. EXAMPLE FOR AN ENCODING PROCESS

A sample string used to demonstrate the algorithm is shown in below chart. The input string is a short list of English words separated by the '/' character. Stepping through the start of the algorithm for this string, you can see that the first pass through the loop, a check is performed to see if the string "/W" is in the table. Since it isn't, the code for '/' is output, and the string "/W" is added to the table. Since we have 256 characters already defined for codes 0-255, the first string definition can be assigned to code 256. After the third letter, 'E', has been read in, the second string code, "WE" is added to the table, and the code for letter 'W' is output. This continues until in the second word, the characters '/' and 'W' are read in, matching string number 256. In this case, the code 256 is output, and a three character string is added to the string table. The process continues until the string is exhausted and all of the codes have been output.

Input String = /WED/WE/WEE/WEB/WET			
Character Input	Code Output	New code value	New String
/W	/	256	/W
E	W	257	WE
D	E	258	ED
/	D	259	D/
WE	256	260	/WE
/	E	261	E/
WEE	260	262	/WEE
/W	261	263	E/W
EB	257	264	WEB
/	B	265	B/
WET	260	266	/WET
EOF	T		

The Compression Process:

The sample output for the string is shown in above chart along with the resulting string table. As can be seen, the string table fills up rapidly, since a new string is added to the table each time a code is output.

III. DECODING

The decoding algorithm works by reading a value from the encoded input and outputting the corresponding string from the initialized dictionary. At the same time it obtains the next value from the input, and adds to the dictionary the concatenation of the string just output and the first character of the string obtained by decoding the next input value.

The decoder then proceeds to the next input value (which was already read in as the "next value" in the previous pass) and repeats the process until there is no more input, at which point the final input value is decoded without any more additions to the dictionary.

In this way the decoder builds up a dictionary which is identical to that used by the encoder, and uses it to decode subsequent input values. Thus the full dictionary does not need be sent with the encoded data; just the initial dictionary containing the single-character strings is sufficient (and is typically defined beforehand within the encoder and decoder rather than being explicitly sent with the encoded data.)

The Decompression Algorithm

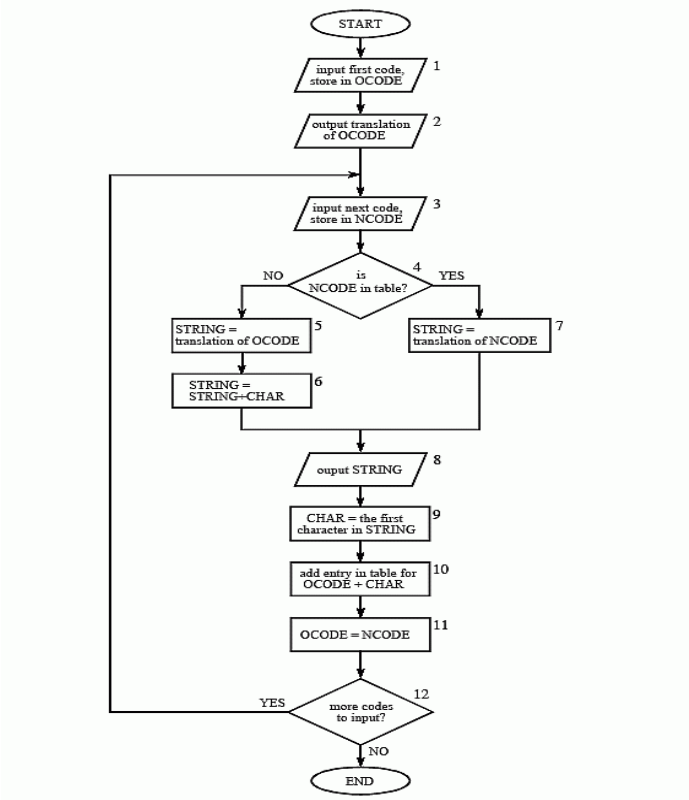


FIGURE 27-8
LZW uncompression flowchart. The variables, *OCODE* and *NCODE* (oldcode and newcode), hold the 12 bit codes from the compressed file, *CHAR* holds a single byte, *STRING* holds a string of bytes.

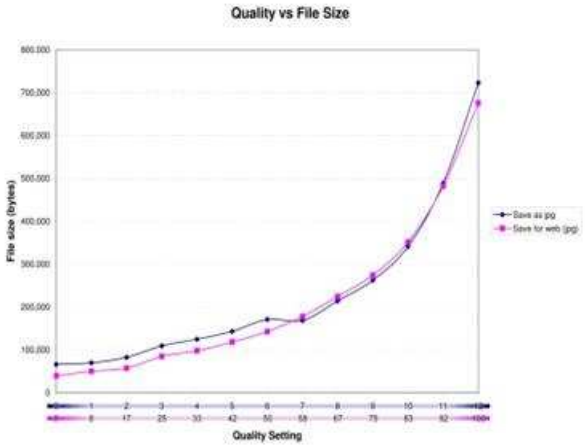
IV. DECOMPRESSION FLOWCHART

AN EXAMPLE FOR DECODING PROCESS

Just like the compression algorithm, it adds a new string to the string table each time it reads in a new code. All it needs to do in addition to that is translate each incoming code into a string and send it to the output. Below chart shows the output of the algorithm given the input created by the compression earlier in the article. The important thing to note is that the string table ends up looking exactly like the table built up during compression. The output string is identical to the input string from the compression algorithm. Note that the first 256 codes are already defined to translate to single character strings, just like in the compression code.

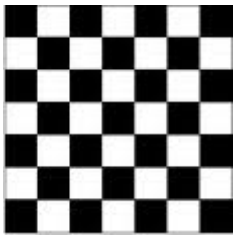
Input Codes: / W E D 256 E 260 261 257 B 260 T				
Input/ NEW_CODE	OLD_CODE	STRING/ Output	CHARACTER	New table entry
/	/	/		
W	/	W	W	256 = /W
E	W	E	E	257 = WE
D	E	D	D	258 = ED
256	D	/W	/	259 = D/
E	256	E	E	260 = /WE
260	E	/WE	/	261 = E/
261	260	E/	E	262 = /WEE
257	261	WE	W	263 = E/W
B	257	B	B	264 = WEB
260	B	/WE	/	265 = B/
T	260	T	T	266 = /WET

IMAGE QUALITY VS IMAGE FILE SIZE GRAPH :



(image quality in x-axis, file size in y-axis)

CHECKER BOARD IMAGE:



LZW compression works best when applied on monochrome images and text files that contain repetitive text/patterns.
For instance, Using LZW compression, a checker board image consisting of repetitive black and white patterns can be compressed upto 70% of its original file size.Thus a high compression ratio can be achieved.

Compression Ratio:

The **compression ratio** expresses the difference between the file size of an uncompressed image, and the file size of the same image when compressed.The compression ratio is equal to the size of the original image divided by the size of the compressed image. This ratio gives an indication of how much compression is achieved for a particular image. Most algorithms have a typical range of compression ratios that they can achieve over a variety of images. Because of this, it is usually more useful to look at an average compression ratio for a particular method.

The compression ratio typically affects the picture quality. Generally, the higher the compression ratio, the poorer the quality of the resulting image. The tradeoff between compression ratio and picture quality is an important one to consider when compressing images.

$$\text{Compression Ratio} = \frac{\text{Size of the original image}}{\text{Size of the compressed image}}$$

Using LZW, 60-70 % of compression ratio can be achieved for monochrome images and text files with repeated data.

Compression/Decompression Speed :

Compression and decompression time is defined as the amount of time required to encode and decode a picture, respectively.

Compression/decompression speed depends on:
- The complexity of the compression algorithm
- The efficiency of the implementation of the algorithm
- The speed of the processor hardware

REFERENCES

[1] The Scientist and Engineer's Guide to Digital Signal Processing by Steven W. Smith
[2] The Data Compression Book by Mark Nelson
[3] Introduction to Data Compression, Third Edition (Morgan Kaufmann Series in Multimedia Information and Systems) by Khalid Sayood
[4] Data Compression in Digital Systems (Digital Multimedia Standards Series) by Roy Hoffman
[5] Compression Algorithms for Real Programmers by Peter Wayner
[6] Data Compression: The Complete Reference by David Salomon