**AA IA2 REPORT**

# Topic: LZW Code Roll no.: 1914078 Batch: B2

LZW compression works by reading a sequence of symbols, grouping the symbols into strings, and converting the strings into codes. Because the codes take up less space than the strings they replace, we get compression.

A typical file data compression algorithm is known as LZW - Lempel, Ziv, Welch encoding. Variants of this algorithm are used in many file compression schemes such as GIF files etc. These are lossless compression algorithms in which no data is lost, and the original file can be entirely reconstructed from the encoded message file.

For example, in images, consecutive scan lines (rows) of the image may be identical. They can then be encoded with a simple code character that represents the lines. In text processing, repetitive words, phrases, and sentences may also be recognized and represented as a code.

The LZW algorithm is a greedy algorithm in that it tries to recognize increasingly longer and longer phrases that are repetitive, and encode them. Each phrase is defined to have a prefix that is equal to a previously encoded phrase plus one additional character in the alphabet. Note: “alphabet” means the set of legal characters in the file. When encoding begins the code table contains only the first 256 entries, with the remainder of the table being blanks. Compression is achieved by using codes 256 through 4095 to represent sequences of bytes.

LZW compression uses a code table, with 4096 as a common choice for the number of table entries. Codes 0-255 in the code table are always assigned to represent single bytes from the input file. In English, for example, the word the occurs more often than any other sequence of three letters, with and, ion, and ing close behind. The new unique symbols are made up of combinations of symbols that occurred previously in the string. It does not always compress well, especially with short, diverse strings. But is good for compressing redundant data, and does not have to save the new dictionary with the data: this method can both compress and decompress data. Although it is impossible to improve on Huffman encoding with any method that assigns a fixed encoding to each character, we can do better by encoding entire sequences of characters with just a few bits. The method of this section takes advantage of frequently occurring character sequences of any length.

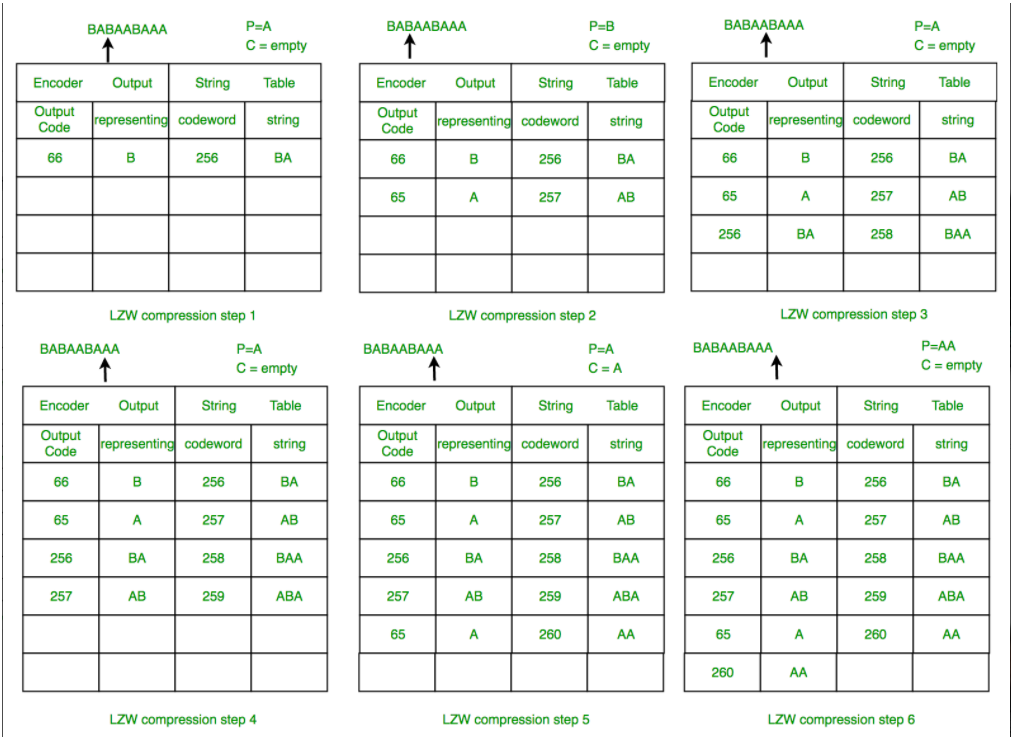
Advantages of LZW over [Huffman](https://www.geeksforgeeks.org/greedy-algorithms-set-3-huffman-coding/) are that LZW requires no prior information about the input data stream. LZW can compress the input stream in one single pass. Another advantage of LZW its simplicity, allowing fast execution.

The algorithm makes use of a dictionary that stores character sequences chosen dynamically from the text. With each character sequence the dictionary associates a number; if s is a character sequence, we use codeword(s) to denote the number assigned to s by the dictionary. The number codeword(s) is called the code or code number of s. All codes have the same length in bits; a typical code size is twelve bits, which permits a maximum dictionary size of 212 = 4096 character sequences.

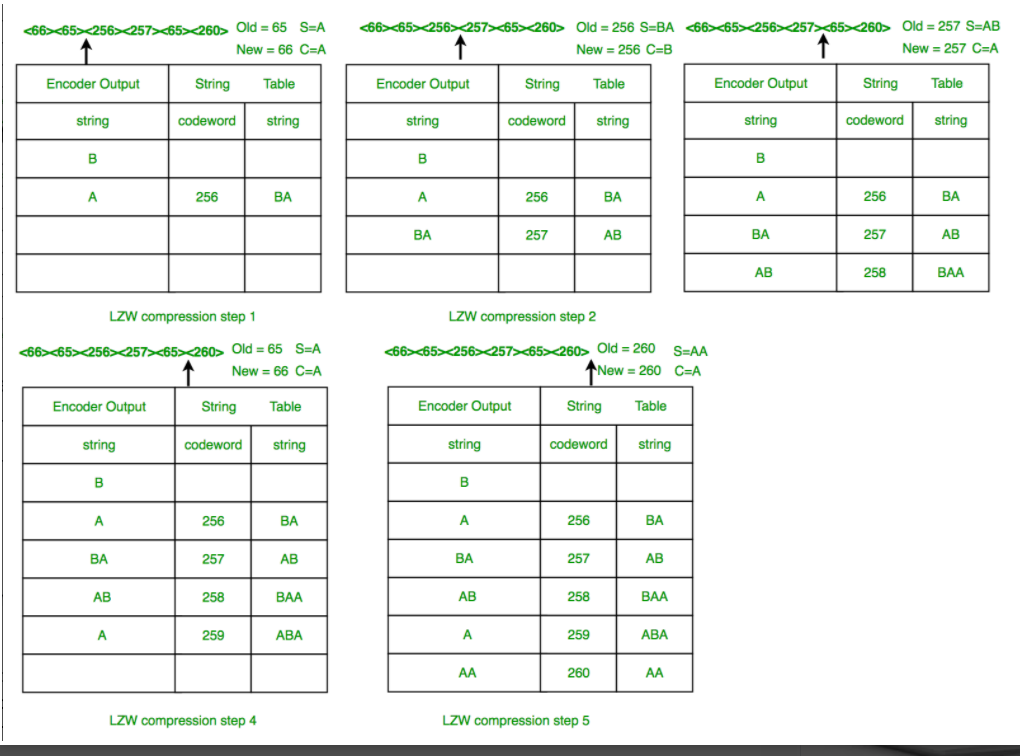
The dictionary is initialized with all possible one-character sequences, that is, the elements of the text alphabet (assume N symbols in the alphabet) are assigned the code numbers 0 through N-1 and all other code numbers are initially unassigned.

**Example:-** String = “**BABAABAAA**”

Compression with algorithm :-



Decompression with algorithm :-



**References:-**

* <https://youtu.be/2FjOJMelZe0>
* <https://www.geeksforgeeks.org/lzw-lempel-ziv-welch-compression-technique/>
* Information theory coding and cryptography, ArijitSaha, Niloptal Manna, SurajitMandal
* Information theory coding and cryptography, Ranjan Bose