[Compression Wizard] [Data Compression Reference Center] [Indexes]

The LZW algorithm

In this algorithm, the same terms are used as in LZ78, with the following addendum:

- A *Root* is a single-character string.

Differences to the LZ78 in the principle of encoding

- Only code words are output. This means that the dictionary cannot be empty at the start: it has to contain all the individual characters (roots) that can occurr in the charstream;
- Since all possible one-character strings are already in the dictionary, each encoding step begins with a one-character prefix, so the first string searched for in the dictionary has two characters;
- The character with which the new prefix starts is the last character of the previous string (C). This is necessary to enable the decoding algorihtm to reconstruct the dictionary without the help of explicit characters in the codestream.

The encoding algorithm

1. At the start, the dictionary contains all possible roots, and P is empty;
2. C := next character in the charstream;
3. Is the string P+C present in the dictionary?
    a. if it is, P := P+C (extend P with C);
    b. if not,
        i. output the code word which denotes P to the codestream;
        ii. add the string P+C to the dictionary;
        iii. P := C (P now contains only the character C);
    c. are there more characters in the charstream?
        - if yes, go back to step 2;
        - if not:
            i. output the code word which denotes P to the codestream;
            ii. END.

Decoding: additional terms

- *Current code word*: the code word currently being processed. It's signified with cW, and the string it denotes with string.cW;
- *Previous code word*: the code word that precedes the current

code word in the codestream. It's signified with pW, and the string it denotes with string.pW.

The principle of decoding

At the start of decoding, the dictionary looks the same as at the start of encoding -- it contains all possible roots.

Let's consider a point in the process of decoding, when the dictionary contains some longer strings. The algorithm remembers the previous code word (pW) and then reads the current code word (cW) from the codestream. It outputs the string.cW, and adds the string.pW extended with the first character of the string.cW to the dictionary. This is the character that would have been explicitly read from the codestream in LZ78. Because of this principle, the decoding algorithm "lags" one step behind the encoding algorithm with the adding of new strings to the dictionary.

A special case occurrs if the cW denotes an empty entry in the dictionary. This can happen because of the explained "lagging" behind the encoding algorithm. It happens if the encoding algorithm reads the string that it has just added to the dictionary in the previous step. During the decoding this string is not yet present in the dictionary. A string can occurr twice in a row in the charstream only if its first and last character are equal, because the next string always starts with the last character of the previous one. This leads to the following decoding rule: the string.pW is extended with its own first character and the resulting string is added to the dictionary and output to the charstream.

The decoding algorithm

1. At the start the dictionary contains all possible roots;
2. cW := the first code word in the codestream (it denotes a root);
3. output the string.cW to the charstream;
4. pW := cW;
5. cW := next code word in the codestream;
6. Is the string.cW present in the dictionary?
    - if it is,
        i. output the string.cW to the charstream;
        ii. P := string.pW;
        iii. C := the first character of the string.cW;
        iv. add the string P+C to the dictionary;
    - if not,
        i. P := string.pW;
        ii. C := the first character of the string.pW;
        iii. output the string P+C to the charstream and add it to the dictionary (now it corresponds to the cW);
7. Are there more code words in the codestream?
    - if yes, go back to step 4;
    - if not, END.

An example

The encoding process is presented in Table 1.

- The column **Step** indicates the number of the encoding step. Each encoding step is completed when the step 3.b. in the encoding algorithm is executed.
- The column **Pos** indicates the current position in the input data.
- The column **Dictionary** shows the string that has been added to the dictionary and its index number in brackets.
- The column **Output** shows the code word output in the corresponding encoding step.

Contents of the dictionary at the beginning of encoding:

(1) A

(2) B

(3) C

Charstream to be encoded:

Pos	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
Char	A	B	B	A	B	A	B	A	C

Table 1: The encoding process

Step	**Pos**	**Dictionary**	**Output**
1.	1	(4) A B	(1)
2.	2	(5) B B	(2)
3.	3	(6) B A	(2)
4.	4	(7) A B A	(4)
5.	6	(8) A B A C	(7)
6.	--	--	(3)

Table 2. explains the decoding process. In each decoding step the algorithm reads one code word (**Code**), outputs the corresponding string (**Output**) and adds a string to the dictionary (**Dictionary**).

Table 2: The decoding process

Step	**Code**	**Output**	**Dictionary**
1.	(1)	A	--
2.	(2)	B	(4) A B
3.	(2)	B	(5) B B
4.	(4)	A B	(6) B A
5.	(7)	A B A	(7) A B A
6.	(3)	C	(8) A B A C

Let's analyze the step **4.** The previous code word (2) is stored in pW, and cW is (4). The string.cW is output ("A B"). The string.pW ("B") is extended with the first character of the string.cW ("A") and the result ("B A") is added to the dictionary with the index (6).

We come to the step **5.** The content of cW=(4) is copied to pW, and the new value for cW is read: (7). This entry in the dictionary is empty. Thus, the string.pW ("A B") is extended with its own

first character ("A") and the result ("A B A") is stored in the dictionary with the index (7). Since cW is (7) as well, this string is also sent to the output.

Practical characteristics
This method is very popular in practice. Its advantage over the LZ77-based algorithms is in the speed because there are not that many string comparisons to perform. Further refinements add variable code word size (depending on the current dictionary size), deleting of the old strings in the dictionary etc. For example, these refinements are used in the GIF image format and in the UNIX `compress` utility for general compression.
Another interesting variation is the LZMW algorithm. It forms a new entry in the dictionary by concatenating the two previous ones. This enables a faster buildup of longer strings.

The LZW method is patented -- the owner of the patent is the Unisys company. It allows free use of the method, except for the producers of commercial software.
