60 LZW (Lempel Ziv Welch)

"All who use swords will be killed with swords."

Every programmer may have the knowledge about data compression. Data compression is the process of reducing the size of the data file. One method of achieving this is by eliminating redundant data. There are many other methods for data compression. In this chapter let's see LZW (Lempel Ziv Welch) algorithm. This algorithm is not much known to people as many books on algorithms ignore this neat algorithm.

60.1 Brief History

In 1977, Abraham Lempel and Jacob Ziv introduced a compression algorithm. Again in 1978, they modified the algorithm and referred it as "dictionary" based compression. The first algorithm was abbreviated as LZ77 and the later as LZ78. Terry Welch altered these algorithms in 1984 and referred the algorithm as LZW. LZW algorithm took its popularity when GIF format used it for compression.

60.2 Principle behind LZW

In LZW compression algorithm, the input file that is to be compressed is read character by character and they are combined to form a string. The process continues till it reaches the end of file. Every new string is assigned some code and stored in *Code table*. They can be referred when the string is repeated with that code. The codes are assigned from 256, since in ASCII character set we have already 256(0-255) characters.

The decompression algorithm expands the compressed file. Here the file, which is created in the compression, is read character by character and it is expanded. This decompression process doesn't require the *Code table* built during the compression.

60.3 LZW Compression

Here the 1st and the 2nd characters are combined to form a string and they are stored in the *Code table*. The code 256(100h) is assigned to the first new string. Then 2nd and 3rd characters are combined and if that string is not available in the *Code table*, it is assigned a new code and it is stored in the *Code table*. Thus we are building a *Code table* with every new string. When the same string is read again, the code already stored in the table will be used. Thus compression occurs when a single code is outputted instead of a set of characters.

The extended ASCII holds only 256(0 to 255) characters and it requires just 8-bits to store each character. But for building the *Code table,* we have to extend the 8-bits to few more bits to hold 256(100h) and above. If we extend it to 12-bits, then we can store up to 4096 elements in the table. So when we store each element in the table it is to be converted to a 12-bit number.

For example, when you want to store `A(dec-65, hex -41)`, `T(dec-84, hex-54)`, `O(dec-79, hex-4F)` and `Z(dec-90, hex-5A)`, you have to store it in bytes as `04, 10, 54, 04, F0, 5A`. The reason is, we have allotted only 12-bits for each character.

Consider a string ‘ATOZOFC’. It takes 7x8(56) bits. Suppose if a code is assigned to it as 400(190h), it will take only 12-bits instead of 56-bits!

60.3.1 Compression Algorithm

```
1.  Specify the number of bits to which you have to extend
2.  read the first character from the file and store it in ch
3.  repeat steps (4) to (7) till there is no character in the file
4.  read the next character and store it in ch2
5.  if ch+ch2 is in the table
                get the code from the table
    otherwise
                output the code for ch+ch2
                add to the table
6.  Store it to the Output file in the specified number of bits
7.  ch = ch2
8.  output the last character ch
9.  exit
```

60.3.2 Example

Input string: ATOZOFCATOZOFCATOZOFC

Characters Read	String Stored / Retrieved	Process in Table	In file
A			Store
T	AT	Store	Store
O	TO	Store	Store
Z	OZ	Store	Store
O	ZO	Store	Store
F	OF	Store	Store
C	FC	Store	Store
A	CA	Store	-
T	AT	Retrieve	Store Relevant Code
O	AT0	Store	-
Z	OZ	Retrieve	Store Relevant Code
O	OZO	Store	-
F	OF	Retrieve	Store Relevant Code
C	OFC	Store	-
A	CA	Retrieve	Store Relevant Code

Characters Read	String Stored / Retrieved	Process in Table	In file
T	CAT	Store	-
O	TO	Retrieve	Store Relevant Code
Z	TOZ	Store	-
O	ZO	Retrieve	Store Relevant Code
F	ZOF	Store	-
C	FC	Retrieve	Store Relevant Code

In this example-string, the first character 'A' is read and then the second character 'T'. Both the characters are concatenated as 'AT' and a code is assigned to it. The code is stored in the *Code table*. Since this is the first string that is new to the table, it is assigned 256(100h). Then the second and the third characters are concatenated to form another new string 'TO'. This string is also new to the *Code table* and the table expands to accommodate this new string and it is assigned the next code 257(101h). Thus whenever a new string is read after concatenation it is assigned a relevant code and the *Code table* is build. The table expands till the code reaches 4096 (since we have assigned 12-bits) or it reaches the end of file.

When the same set of characters that is stored in the table is again read it is assigned to the code in the *Code table*. Thus according to the number of bits specified by the program the output code is stored. In other words, if we have extended the bits from 8 to 12 then the characters that is stored in 8-bits should be adjusted so as to store it in 12-bit format.

60.4 LZW Decompression

The file that is compressed is read byte by byte. The bytes are concatenated according to the number of bits specified by us. For example, we have used 12-bits for storing the elements so we have to read first 2-bytes and get the first 12-bits from that 16-bits. Using this bits *Code table* is build again without the *Code table* previously created during the compression. Use the remaining 4-bits from the previous 2-bytes and next byte to form the next code in the string table. Thus we can build the *Code table* and use it for decompression.

This decompression algorithm builds its own *Code table* and it will be same as the table created during the compression. The decompression algorithm refers this newly created *Code table* but not the *Code table* created during the compression. This is the main advantage in this algorithm.

60.4.1 Decompression Algorithm

1. read the character l
2. convert l to its original form
3. output l
4. repeat steps(5) to (10) till there is no character in the file
5. read a character z
6. convert l+z to its original form
7. output in character form

```
8.  if l+z is new then
            store in the code table
9.  add l+z first char of entry to the code table
10. l = first char of entry
11. exit
```

60.4.2 Example

Consider the same example given above and do the decompression.

Compressed Bytes (in hex)	Strings given after converting from 12-bit format to 8-bit format
04	
10	→ A, T
84	
04	
F0	→ O, Z
5A	
04	
F0	→ O, F
46	
04	
31	→ C, AT
00	
10	
21	→ OZ, OF
04	
10	
61	→ CA, TO
01	
10	
31	→ ZO, FC
05	

Here each byte is read one by one as hexadecimal code and 3 of the bytes are combined so as to convert them from a 12-bit format to a 8-bit character (ASCII) format.

Thus the bytes 04, 10 & 84 are combined as 041084. The combined code is split to get A(041) and T(084). The table is also built concurrently when each new string is read. When we read 100, 102 etc., we can refer to the relevant code in the table and output the relevant code to the file. For example, when we reach the 4th set of characters and read 04, 31 and 00 they must be converted to 12-bit form as 043 and 100 will refer to the code in the table and outputs the string C and AT respectively. Thus we can get all the characters without knowing the previous *Code table*.

Suggested Projects

1. Write your own compression utility using LZW algorithm.