Enhanced LZW Algorithm with Less Compression Ratio

Amit Setia and Priyanka Ahlawat

Department of Computer Engineering
National Institute of Technology, Kurukshetra, India
{amitsetia50,mannpammy}@gmail.com

Abstract. LZW is the one known compression algorithm appropriate for communication. LZW data compression algorithm is popular for data compression because it is an adaptive algorithm and achieves an excellent compromise between compression performance and speed of execution. LZW is a dictionary based data compression algorithm, which compress the data in a lossless manner so that no information is lost. LZW algorithm requires no extra communication from the encoder to the decoder. In this paper we present a scheme to eliminate the spaces from the data so that high compression factor achieves.

Keywords: compression, adaptive, encoding, decoding.

1 Introduction

LZW [Welch 1984] is a popular variant of LZ78 [Ziv and Lempel 1978], developed by Terry Welch in 1984. It is a dictionary based adaptive algorithm. The first 256 entries are occupied in the dictionary before any data is input. Most of the data file to which we want to compress contains many spaces. By eliminate these spaces from the data file, results in high compression factor or less compression ratio.

The rest of the paper is organized as follows. Section 2 reviews the basic concepts of compression and encryption. Section 3 presents the compression techniques. Section 4 describes the proposed system. Section 5 deals with results. Section 6 concludes the paper. Section 7 presents the future work.

2 Basic Concepts

There are various techniques to compression.

Compression Techniques:

a) LZW compression
b) Huffman coding
c) Arithmetic coding
d) Run length encoding
e) LZ 77
f) LZ 78 and so on.

In our system we are using LZW technique.

Aswatha Kumar M. et al. (Eds.): Proceedings of ICAdC, AISC 174, pp. 347–351.
springerlink.com

3 Compression Technique

3.1 LZW Compression

It is a dictionary based algorithm. It is a variant of Lempel Ziv 78 compression algorithm. In this we initialize the dictionary to all symbols in alphabet. In common case of 8-bit symbols, first 256 entries of the dictionary 0 through 255 are occupied before any data is input. LZW token consist a pointer to the dictionary. When the dictionary is initialized, the next input character finds in dictionary.

The idea of LZW is that the encoder input symbols one by one and accumulates them in string I. After each symbol is input and is concatenated to I, dictionary is searched for string I. As long as string I is found in dictionary, the process continues. But at some point if adding text symbol suppose x causes the search to fail, string I is in dictionary but string Ix is not. At this point the encoder outputs the dictionary pointer that points to string I, and saves string Ix in the next available entry in the dictionary and then initialize string I to symbol x.

3.1.1 LZW Encoding Algorithm

Algorithm:

Step 1 Initialize dictionary to contain single character string.
Step 2 Read first input character prefix string ω from the data file.
Step 3 Read next input character k from the data file.
a) If no such k (input exhausted): output:=code(ω):Then EXIT
b) If ωk exists in dictionary: ω: = ωk; Repeat Step 3.
c) Else If ωk not in dictionary. Then output :=code (ω)
Dictionary: = ωk;
ω :=k;
Repeat step 3.
Step 4 End

3.1.2 LZW Decoding Algorithm

Algorithm:

Step 1 Read first input code and CODE=OLD code=input code with CODE=code (k), output=k, Fin char=k.
Step 2 Read next input code, CODE =INCODE=next input code.
If no new code: EXIT. Else:
Step 3 If CODE=code (ωk): stack = k
CODE: =code (ω)
Repeat Step 3
Else if CODE = code (k): output=k, Fin char=k.
Do while stack not empty:
Output = Stack top, Pop stack.
Dictionary=OLD code, k.
OLD code=IN code;
Repeat step 2.
Step 4 End

3.2 Advantages of LZW Algorithm

1. LZW data compression algorithm is an adaptive and very effective means to save storage space and network bandwidth.
2. LZW algorithm is easy to implement.
3. It is a lossless compression algorithm, so no loss of information is there.

4 Proposed Scheme

In LZW algorithm, dictionary size is determined by the general unary code.

Table 1.

n	a=(3+n*2)	nth codeword	no. of code words	Range of integers
0	3	0xxx	2^3 =8	0-7
1	5	10xxxxx	2^5 =32	8-39
2	7	110xxxxxxx	2^7 =128	40-167
3	9	111xxxxxxxxx	2^9 =512	168-679
		Total	680	

Now in LZW algorithm if we make dictionary of 680 entries. Each entry occupy 9 bits up to 680 entries, And if we want to increase dictionary size up to 2044 entries then each entry occupy 10 bits.

But in our proposed scheme we eliminate the spaces from the input data file. In our scheme if we make dictionary of 680 entries, Each entry occupy 10 bits, i.e. 1 bit higher than those in case of LZW encoding algorithm, 9bit data plus one parity bit used for checking the next character is space or not. If parity bit is set then next character is space otherwise not. But we save the 9 bits which is used for space. Now there are number of spaces in data file say n, Now we save n*9 bit space. But some extra parity bits are also sent, but many words in data file have small length. So spaces between these words occupy lot of space, so by eliminate these spaces we can achieve high compression.

4.1 Enhanced LZW Encoding Algorithm

Algorithm:

Step 1 Initialize dictionary to contain all 0 to 255 single character string.
Step 2 Read first input character prefix string ω from the input data file.

Step 3 Read next input character says k from the input data file.
a) If no such k (input exhausted): output:=code(ω):Then EXIT
b) If k=space, then set M.S.B bit of ω equals to 1, Repeat step 3.
c) If ωk exists in dictionary: ω: = ωk; Repeat Step 3.
d) Else If ωk not in dictionary. Then output :=code (ω)
Dictionary: = ωk;
ω :=k;
Repeat step 3.
Step 4 End

4.1.1 Enhanced LZW Encoding Algorithm Description

In step no. 3 (b) we set the parity bit or M.S.B bit of suffix ω. that helps in decoding site.

4.2 Enhanced LZW Decoding Algorithm

Algorithm:

Step 1 Read first input character ω, left shift the input character ω by 1. ie ω <<1. And result is stored in the carry bit. Then right shift the input character ω by 1. ie ω >>1. And CODE=OLD code=input code with CODE=code (k), output=k, Fin char=k.
Step 2 If carry bit = 1
Then next input character k = space.
Go to step 3.
Else Read next input character k from the input data file, left shift the input character k by 1. ie k <<1. And result is stored in the carry bit. Then right shift the input character k by 1. ie k >>1.
CODE = INCODE = next input code
If no new code: EXIT. Else:
Step 3 If CODE=code (ωk): stack = k
CODE: =code (ω)
Repeat Step 3
Else if CODE = code (k): output=k, Fin char=k.
Do while stack not empty:
Output = Stack top, Pop stack.
Dictionary = OLD code, k.
OLD code = IN code;
Repeat step 2.
Step 4 End

4.2.1 Enhanced LZW Decoding Algorithm Description

In step 1 we first left shift the character ω by 1 and result stored in carry bit. And now for retrieving actual data we right shift the character by 1.

5 Results

Table 2.

File size	Compression ratio with LZW compression	Compression ratio with enhanced LZW compression
5.7 kb	0.50	0.47
10 kb	0.61	0.53

We have taken two files one is of 5.7 kb and another is of 10 kb. When we compressed first file which is of 5.7 kb using LZW, the compression ratio of 0.50 achieved. But when we compressed same file using enhanced LZW then we achieved compression ratio of 0.47. In second case when we compressed another file of 10 kb size using LZW algorithm then we achieved compression ratio of 0.61. But when we compressed same file using enhanced LZW we achieved compression ratio of 0.50.

But compression ratios in case of enhanced LZW vary data to data.

6 Conclusion

The work present in this paper can be summarized as:

In this paper we accomplished a system which eliminate spaces from the data file so that we can achieve high compression.

7 Future Work

The proposed system can be combined with cryptography and steganography techniques for more security, which encrypt and hide the existence of the information.

References

[1] Jiang, J.: Pipeline algorithm of RSA data encryption and data compression. In: IEEE Proceeding of International Conference on Communication Technology, vol. 2, pp. 1088–1091 (1996)

[2] Kruse, H., Mukherjee, A.: Data compression using text encryption. In: IEEE Data Compression Conference, p. 447 (1997)

[3] Lin, M.B., Chang, Y.Y.: A new architecture of a two-stage lossless data compression and decompression algorithm. IEEE Transaction on VLSI Systems 17, 1297–1303 (2009)

[4] Welch, T.: A technique for high performance data compression. IEEE Computer 17, 8–19 (1984)

[5] Ziv, J., Lempel, A.: A universal algorithm for sequential data compression. IEEE Transaction on Information Theory 23, 337–343 (1977)

[6] Ziv, J., Lempel, A.: Compression of individual sequences via variable length coding. IEEE Transaction on Information Theory 24, 530–536 (1978)