LZW Lossless Text Data Compression Algorithm – A Review

Ezhilarasu P

Associate Professor, Department of Computer Science and Engineering
Hindusthan College of Engineering and Technology, Coimbatore, India.
prof.p.ezhilarasu@gmail.com

Karthik Kumar P

Research Scholar
PSG College of Technology, Coimbatore, India

Abstract: In this paper, we discuss LZW data compression techniques for strings with various conditions. Initially, string contains the single character with the varied string length of 10, 20, 30, 40, 50, and 100 taken. Then alternate characters and finally the mixed combination of characters taken for the compression. Its compression ratio, space savings also calculated. Each condition compared with other conditions.

Keywords: LZW, Compression, Encoder, Decoder.

I. INTRODUCTION

The system of reducing the size of a data file referred to as data compression [1]. Data compression involves the tradeoff called space-time complexity. If the data stored as it is, then there is no need to compress and decompress the data. We need vast amount of storage for that. However, in many situations there is a need for applying resource management techniques. In compression techniques, there is a need for managing the storage efficiently.

Because of fewer amounts of data, transfer of data from source to destination can be performed with less amount of time. For instance, if the data size is 50MB and the transfer rate between source and destination is 25 kbps. The time need for the transfer can be calculated by the equation 1.

Time for transfer = Input data / transfer rate (1)

1 MB = 1024 KB and 1 KB = 8 kb

Input data = 50 MB

= 50 * 1024 KB

= 50 * 1024 * 8 kb

Transfer rate = 25 kbps

So time taken for transfer = (50 * 1024 * 8) / 25

= 2 * 1024 * 8

= 16384 seconds

If the given data compressed into 20MB, then the time taken for transfer will be 6553.6 seconds.

If the allowed storage for destination machine is 40GB, then the target machine can store the following number of files by using the equation 2.

Number of files can be stored = Total amount of storage / Size of the file (2)

1GB = 1024 MB

= 40 GB / 50 MB

= 40 * 1024 MB / 50 MB

=819.2 files

So destination system can store 819 files for uncompressed data.

For compressed file, it can store

= 40 * 1024 MB / 20 MB

= 2 * 1024

= 2048 files.

The space and time complexity based on compression ratio. It can be calculated by using the following equation 3.

Compression ratio = Actual data / Compressed Data (3)

= 50 MB / 20 MB

= 2.5

The compressed data takes less storage with faster transfer rate than original data.

In the file, we have many types. It may be a text file, image file, audio or video file. The compression ratio differs for each file types.

The data compression also has some limitations. For compressing the video files of vast size sometimes, we need special hardware. For compression and decompression, we need some amount of time. In some time, the time may be more. During compression and decompression, the some data may be lost. The limitations summed as

1. Processing cost
2. Processing time
3. Quality of data

In compression, we have the following types

1. Lossy compression (Destination data size is less than source data)
2. Lossless compression (Destination data size is equal to source data).

In this paper, we discuss LZW lossless data compression algorithm. Lempel–Ziv–Welch (LZW) is a universal lossless data compression algorithm. It created by Abraham Lempel, Jacob Ziv, and Terry Welch. It was published by Welch in 1984 as an improved implementation of the LZ78 algorithm published by Lempel and Ziv in 1978 [2].

II. RELATED WORK

Ziv J and Lempel A [1977] proposed a universal algorithm for sequential data compression [3]. Then after a year [1978] they proposed a compression method about the cCompression of individual sequences using variable-rate coding [4]. Bell T, Witten I and Cleary J [1990] discussed lossless compression. It focuses on text compression and language modeling. It contains numerous statistical studies on text compression [5].

Mark Nelson and Jean-loup Gailly [1995] explained the basics of data compression algorithms and classified the compression area. It includes lossless and lossy algorithms, the modeling-coding paradigm and statistical and dictionary schemes [6].David Salomon [2000] described many different compression algorithms together with their benefits, disadvantages, and common usages. He gave a broad overview on lossless and lossy compression [7].

Khalid Sayood [2000] gave an introduction into the wide field of coding algorithms, both lossless and lossy, with mathematical and theoretical background information [8].Ross Williams [1991] described lossless compression algorithms based on Markov models [9]. Ian Witten, Alistair Moffat and Timothy Bell [1999] gave an introduction about information retrieval. They also emphasized on indexing, querying and implementation aspects mostly based on lossless compression [10]. Many books on data compression [11, 12, 13, 14, 15] and research paper on LZW compression [16] also described in detail about data compression and LZW compression.

III. LZW ENCODING ALGORITHM

Initialize Dictionary by using with 256 ASCII codes for representing 256 characters; values are from 0 – 255.

1. Initialize codeword as 255 and starting input character as first character of the given input.
2. If not the end of the input, Suffix the input. If the end of the input then go to step 6.
3. Check the input character(s). If available in the dictionary then go to step 2.
4. Increment codeword by one then assign that value to the collection of characters.
5. Take the immediate input character after the codeword then go to step 2.
6. Stop the process.

IV. LZW DECODING ALGORITHM

The LZW decompressor produces the same string table during decompression. It is the reversal of LZW encoding algorithm.

A.LZW ENCODING

TABLE I. FOR STRING LENGTH 10 AND SINGLE CHARACTER

EEEEEEEEEE(LENGTH 10)

OUTPUT	DICTIONARY	
	CODE WORD	STRING
69	256	EE
256	257	EEE
257	258	EEEE
258		EEEE(REMAINING)

The string length is 10
Actual space needed = 10 * 8 = 80 bits
AFTER ENCODING
Space needed = 4 * 12 = 48 bits

TABLE II. FOR STRING LENGTH 20 AND SINGLE CHARACTER

EEEEEEEEEEEEEEEEEEEE(LENGTH 20)

OUTPUT	DICTIONARY	
	CODE WORD	STRING
69	256	EE
256	257	EEE
257	258	EEEE
258	259	EEEEE
259	260	EEEEEE
259		EEEEE(REMAINING)

The string length is 20
Actual space needed = 20 * 8 = 160 bits
AFTER ENCODING
Space needed = 6 * 12 = 72 bits

TABLE III. FOR STRING LENGTH 30 AND SINGLE CHARACTER

EEEEEEEEEEEEEEEEEEEEEEEEEEEEEE(LENGTH 30)

OUTPUT	DICTIONARY	
	CODE WORD	STRING
69	256	EE
256	257	EEE
257	258	EEEE
258	259	EEEEE
259	260	EEEEEE
260	261	EEEEEEE
261	262	EEEEEEEE
256		EE(REMAINING)

The string length is 30
Actual space needed = 30 * 8 = 240 bits
AFTER ENCODING
Space needed = 8 * 12= 96 bits

TABLE IV. FOR STRING LENGTH 40 AND SINGLE CHARACTER

EEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEE(LENGTH 40)

OUTPUT	DICTIONARY	
	CODE WORD	STRING
69	256	EE
256	257	EEE
257	258	EEEE
258	259	EEEEE
259	260	EEEEEE
260	261	EEEEEEE
261	262	EEEEEEEE
262	263	EEEEEEEEE
258		EEEE(REMAINING)

The string length is 40
Actual space needed = 40 * 8 = 320 bits
AFTER ENCODING
Space needed = 9 * 12= 108 bits

TABLE V. FOR STRING LENGTH 50 AND SINGLE CHARACTER

EEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEE(LENGTH 50)

OUTPUT	DICTIONARY	
	CODE WORD	STRING
69	256	EE
256	257	EEE
257	258	EEEE
258	259	EEEEE
259	260	EEEEEE
260	261	EEEEEEE
261	262	EEEEEEEE
262	263	EEEEEEEEE
263	264	EEEEEEEEEE
259		EEEEE(REMAINING)

The string length is 40
Actual space needed = 50 * 8 = 400 bits
AFTER ENCODING
Space needed = 10 * 12= 120 bits

TABLE VI. FOR STRING LENGTH 100 AND SINGLE CHARACTER

EEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEE
EEEEEEEEEEEEEEEEEEEEEEEEEEE(LENGTH 100)

OUTPUT	DICTIONARY	
	CODE WORD	STRING
69	256	EE
256	257	EEE
257	258	EEEE
258	259	EEEEE
259	260	EEEEEE
260	261	EEEEEEE
261	262	EEEEEEEE
262	263	EEEEEEEEE
263	264	EEEEEEEEEE
264	265	EEEEEEEEEEE
265	266	EEEEEEEEEEEE
266	267	EEEEEEEEEEEEE
267	268	EEEEEEEEEEEEEE
263		EEEEEEEEE(REMAINING)

The string length is 100
Actual space needed = 100 * 8 = 800 bits
AFTER ENCODING
Space needed = 14 * 12= 168 bits

TABLE VII. FOR STRING LENGTH 10 AND ALTERNATE CHARACTER

EFEFEFEFEF(LENGTH 10)

OUTPUT	DICTIONARY	
	CODE WORD	STRING
69	256	EF
70	257	FE
256	258	EFE
258	259	EFEF
257	260	FEF
70		F(REMAINING)

The string length is 10
Actual space needed = 10 * 8 = 80 bits
AFTER ENCODING
Space needed = 6 * 12 = 72 bits

TABLE VIII. FOR STRING LENGTH 20 AND ALTERNATE CHARACTER

EFEFEFEFEFEFEFEFEFEF(LENGTH 20)

OUTPUT	DICTIONARY	
	CODE WORD	STRING
69	256	EF
70	257	FE
256	258	EFE
258	259	EFEF
257	260	FEF
260	261	FEFE
259	262	EFEFE
259		EFEF(REMAINING)

The string length is 20
Actual space needed = 20 * 8 = 160 bits
AFTER ENCODING
Space needed = 8 * 12 = 96 bits

TABLE IX. FOR STRING LENGTH 30 AND ALTERNATE CHARACTER

EFEFEFEFEFEFEFEFEFEFEFEFEFEFEF(LENGTH 30)

OUTPUT	DICTIONARY	
	CODE WORD	STRING
69	256	EF
70	257	FE
256	258	EFE
258	259	EFEF
257	260	FEF
260	261	FEFE
259	262	EFEFE
262	263	EFEFEF
261	264	FEFEF
264		FEFEF(REMAINING)

The string length is 30
Actual space needed = 30 * 8 = 240 bits
AFTER ENCODING
Space needed = 10 * 12 = 120 bits

TABLE X. FOR STRING LENGTH 40 AND ALTERNATE CHARACTER

EFEFEFEFEFEFEFEFEFEFEFEFEFEFEFEFEFEFEFEF(LENGTH 40)

OUTPUT	DICTIONARY	
	CODE WORD	STRING
69	256	EF
70	257	FE
256	258	EFE
258	259	EFEF
257	260	FEF
260	261	FEFE
259	262	EFEFE
262	263	EFEFEF
261	264	FEFEF
264	265	FEFEFE
263	266	EFEFEFE
259		EFEF(REMAINING)

The string length is 40
Actual space needed = 40 * 8 = 320 bits
AFTER ENCODING
Space needed = 12 * 12 = 144 bits

TABLE XI. FOR STRING LENGTH 50 AND ALTERNATE CHARACTER

EFEFEFEFEFEFEFEFEFEFEFEFEFEFEFEFEFEFEFEFEFEFEFEFEF(LENGTH 50)

OUTPUT	DICTIONARY	
	CODE WORD	STRING
69	256	EF
70	257	FE
256	258	EFE
258	259	EFEF
257	260	FEF
260	261	FEFE
259	262	EFEFE
262	263	EFEFEF
261	264	FEFEF
264	265	FEFEFE
263	266	EFEFEFE
266	267	EFEFEFEF
265	268	FEFEFEF
70		F(REMAINING)

The string length is 50
Actual space needed = 50 * 8 = 400 bits
AFTER ENCODING
Space needed = 14 * 12 = 168 bits

TABLE XII. FOR STRING LENGTH 100 AND ALTERNATE CHARACTER

EFEFEFEFEFEFEFEFEFEFEFEFEFEFEFEFEFEFEFEFEFEFEFEFEFEFEFEFEFEFEFEFEFEFEFEFEFEFE
FEFEFEFEFEFEFEFEFEFEFEF(LENGTH 100)

OUTPUT	DICTIONARY	
	CODE WORD	STRING
69	256	EF
70	257	FE
256	258	EFE
258	259	EFEF
257	260	FEF
260	261	FEFE
259	262	EFEFE
262	263	EFEFEF
261	264	FEFEF
264	265	FEFEFE
263	266	EFEFEFE
266	267	EFEFEFEF
265	268	FEFEFEF
268	269	FEFEFEFE
267	270	EFEFEFEFE
270	271	EFEFEFEFEF
269	272	FEFEFEFEF
272	273	FEFEFEFEFE
271		EFEFEFEFEF(REMAINING)

The string length is 100
Actual space needed = 100 * 8 = 800 bits
AFTER ENCODING
Space needed = 19 * 12 = 228 bits

TABLE XIII. FOR STRING LENGTH 10 AND MIXED CHARACTER

EFGEEEGFGE(LENGTH 10)

OUTPUT	DICTIONARY	
	CODE WORD	STRING
69	256	EF
70	257	FG
71	258	GE
69	259	EE
259	260	EEG
71	261	GF
257	262	FGE
69		E(REMAINING)

The string length is 10
Actual space needed = 10 * 8 = 80 bits
AFTER ENCODING
Space needed = 8 * 12 = 96 bits

TABLE XIV. FOR STRING LENGTH 20 AND MIXED CHARACTER

FGGEEFEEGEFFGEGGGGEF(LENGTH 20)

OUTPUT	DICTIONARY	
	CODE WORD	STRING
70	256	FG
71	257	GG
71	258	GE
69	259	EE
69	260	EF
70	261	FE
259	262	EEG
258	263	GEF
70	264	FF
256	265	FGE
69	266	EG
257	267	GGG
257	268	GGE
260		EF(REMAINING)

The string length is 20
Actual space needed = 20 * 8 = 160 bits
AFTER ENCODING
Space needed = 14 * 12 = 168 bits

TABLE XV. FOR STRING LENGTH 30 AND MIXED CHARACTER

FGGEEFEEGEEFGEEEGFGEFFGEGGGGEF(LENGTH 30)

OUTPUT	DICTIONARY	
	CODE WORD	STRING
70	256	FG
71	257	GG
71	258	GE
69	259	EE
69	260	EF
70	261	FE
259	262	EEG
258	263	GEE
260	264	EFG
263	265	GEEE
69	266	EG
71	267	GF
256	268	FGE
260	269	EFF
268	270	FGEG
257	271	GGG
257	272	GGE
260		EF(REMAINING)

The string length is 30
Actual space needed = 30 * 8 = 240 bits
AFTER ENCODING
Space needed = 18 * 12 = 216 bits

TABLE XVI. FOR STRING LENGTH 40 AND MIXED CHARACTER

FGGEEFEEGEEFGEEEGFGEFFGEGGGGEFFGGEEFEEGE (LENGTH 40)

OUTPUT	DICTIONARY	
	CODE WORD	STRING
70	256	FG
71	257	GG
71	258	GE
69	259	EE
69	260	EF
70	261	FE
259	262	EEG
258	263	GEE
260	264	EFG
263	265	GEEE
69	266	EG
71	267	GF
256	268	FGE
260	269	EFF
268	270	FGEG
257	271	GGG
257	272	GGE
269	273	EFFG
272	274	GGEE
260	275	EFE
262	276	EEGE
69		E(REMAINING)

The string length is 40

Actual space needed = 40 * 8 = 320 bits

AFTER ENCODING

Space needed = 22 * 12 = 264 bits

TABLE XVII. FOR STRING LENGTH 50 AND MIXED CHARACTER

FGGEEFEEGEEFGEEEGFGEFFGEGGGGEFFGGEEFEEGEFFGEGGGGEF (LENGTH 50)

OUTPUT	DICTIONARY	
	CODE WORD	STRING
70	256	FG
71	257	GG
71	258	GE
69	259	EE
69	260	EF
70	261	FE
259	262	EEG
258	263	GEE
260	264	EFG
263	265	GEEE
69	266	EG
71	267	GF
256	268	FGE
260	269	EFF
268	270	FGEG
257	271	GGG
257	272	GGE
269	273	EFFG
272	274	GGEE
260	275	EFE
262	276	EEGE
273	277	EFFGE
266	278	EGG
271	279	GGGE
260		EF(REMAINING)

The string length is 50

Actual space needed = 50 * 8 = 400 bits

AFTER ENCODING

Space needed = 25 * 12 = 300 bits

TABLE XVIII. FOR STRING LENGTH 100 AND MIXED CHARACTER

FGGEEFEEGEEFGEEEGFGEFFGEGGGGEFFGGEEFEEGEFFGEGGGGEFFGGEEFEEGEEFGEEEGFGE FFGEGGGGEFFGGEEFEEGEFFGEGGGGEF(LENGTH 100)

OUTPUT	DICTIONARY	
	CODE WORD	STRING
70	256	FG
71	257	GG
71	258	GE
69	259	EE
69	260	EF
70	261	FE
259	262	EEG
258	263	GEE
260	264	EFG
263	265	GEEE
69	266	EG
71	267	GF
256	268	FGE
260	269	EFF
268	270	FGEG
257	271	GGG
257	272	GGE
269	273	EFFG
272	274	GGEE
260	275	EFE
262	276	EEGE
273	277	EFFGE
266	278	EGG
271	279	GGGE
273	280	EFFGG
263	281	GEEF
261	282	FEE
266	283	EGE
259	284	EEF
268	285	FGEE
262	286	EEGF
268	287	FGEF
70	288	FF
270	289	FGEGG
279	290	GGGEF
288	291	FFG
274	292	GGEEF
282	293	FEEG
258	294	GEF
291	295	FFGE
278	296	EGGG
272	297	GGEF
70		F(REMAINING)

The string length is 100
Actual space needed = 100 * 8 = 800 bits

AFTER ENCODING
Space needed = 43 * 12 = 516 bits

V. LZW DECODING

The table 1 shows LZW encoding for the input EEEEEEEEEE. The output is 69,256,257,258. The decoding is done by using codeword value. The output 69 replaced by the codeword E, 256 by EE, 257 by EEE and 258 BY EEEE. The obtained results combined as E, EE, EEE, EEEE. The resultant output string will be EEEEEEEEEE, which is similar to given input EEEEEEEEEE. This process used in table 2-18. The resultant output same as that of input.

VI. RESULTS AND DISCUSSIONS

The compression ratio and space savings derived for three cases namely single character, alternate character and mixed character. It tested for string of different length as 10,20,30,40,50 and 100. The results are given in the table 19 & 20 and figure 1 & 2.

TABLE XIX. COMPRESSION RATIO FOR SINGLE, ALTERNATE AND MIXED CHARACTERS

COMPRESSION RATIO

S.NO	CHARACTER COMBINATION	STRING LENGTH					
		10	20	30	40	50	100
1	SINGLE	1.66:1	2.22:1	2.5:1	2.96:1	3.33:1	4.76:1
2	ALTERNATE	1.11:1	1.66:1	2:1	2.22:1	2.38:1	3.51:1
3	MIXED	0.83:1	0.95:1	1.11:1	1.21:1	1.33:1	1.55:1

TABLE XX. SPACE SAVINGS FOR SINGLE, ALTERNATE AND MIXED CHARACTERS

SPACE SAVINGS

S.NO	CHARACTER COMBINATION	STRING LENGTH					
		10	20	30	40	50	100
1	SINGLE	39.75%	54.95%	60%	66.22%	69.97%	78.99%
2	ALTERNATE	9.91%	39.75%	50%	54.95%	57.98%	71.51%
3	MIXED	-20.48%	-5.26%	9.91%	17.35%	24.81%	35.48

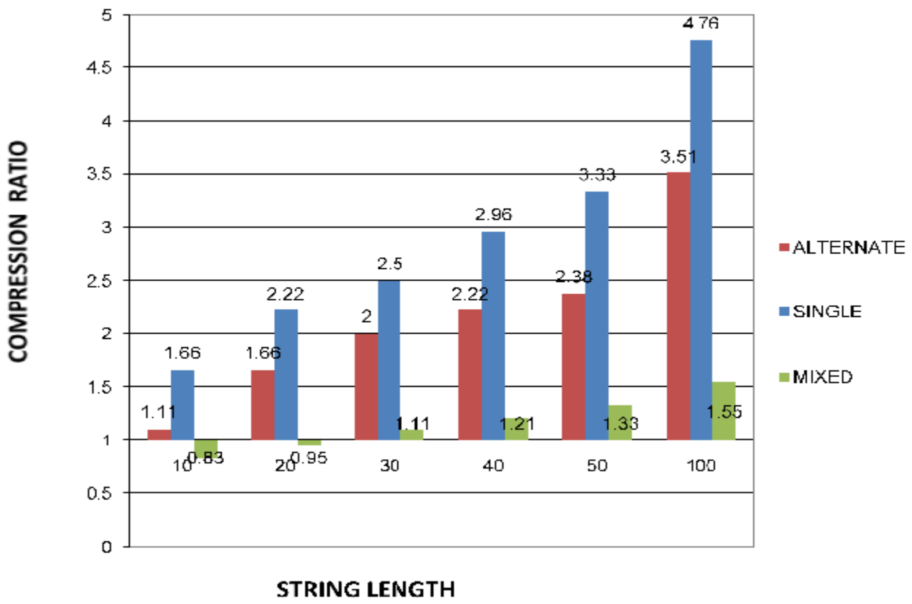

Fig. 1. Compression ratio for single, alternate and mixed characters

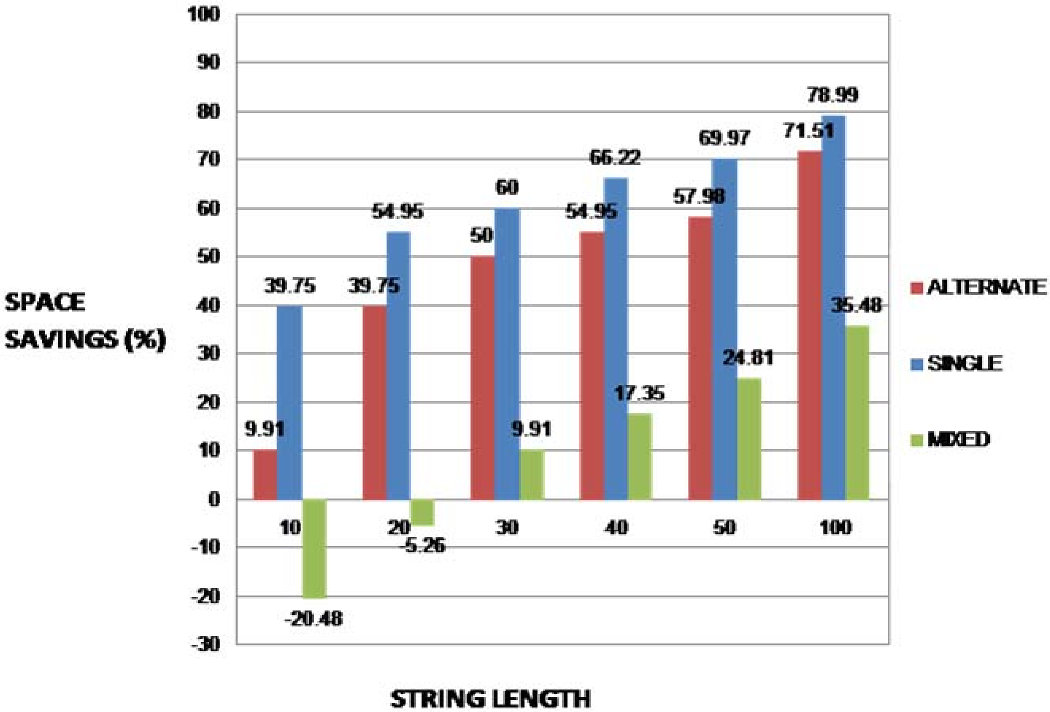

Fig. 2. Space savings for single, alternate and mixed characters

VII. CONCLUSION

The obtained results show that LZW algorithm provides better compression ratio and space savings as string length increases for single, alternate and mixed character. This paper will help the students to understand the data compression using LZW algorithm. The prior knowledge about data compression is also not needed for understanding this paper.

VIII. REFERENCE

[1] Available online at: http://en.wikipedia.org/wiki/Data_compression
[2] Available online at: http://en.wikipedia.org/wiki/Lempel- Ziv-Welch
[3] Ziv, J., and Lempel, A., “A universal algorithm for sequential data compression,” IEEE Transactions on Information Theory, Volume 23, Number 3, May 1977, pages 337-343.
[4] Ziv, J., and Lempel, A., "Compression of individual sequences via variable-rate coding," IEEE Transactions on Information Theory, Volume 24, Number 5, September 1978, pages 530-536.
[5] Timothy Bell, John Cleary and Ian Witten [1990],” Text Compression”, Prentice-Hall, Englewood, United States of America, 318 pages
[6] Mark Nelson and Jean-loup Gailly [1995], “The Data Compression Book”, M&T Books, New York, United States of America, 2nd edition, 541 pages
[7] David Salomon [2000], “Data Compression: The Complete Reference” Springer, New York, Berlin, Heidelberg, United States of America, Germany, 2nd edition, 823 pages
[8] Khalid Sayood [2000], “Introduction to Data Compression”, Morgan Kaufmann Publishers, Burlington, United States of America, 2nd edition, 600 pages
[9] Ross Williams [1991], “Adaptive Data Compression”, Kluwer Books, Norwell, United States of America, 382 pages
[10] Ian Witten, Alistair Moffat and Timothy Bell [1999],” Managing Gigabytes: Compressing and Indexing Documents and Images”, Morgan Kaufmann Publishing, San Francisco, United States of America, 2nd edition, 519 pages
[11] David Salomon and Giovanni Motta [2010], “Handbook of Data Compression”, Springer London
[12] Storer, J.A., (1988) Data Compression , Computer Science Press, Rockville, MD
[13] Blelloch, E., 2002. Introduction to Data Compression, Computer Science Department, Carnegie Mellon University.
[14] Lynch, Thomas J., Data Compression: Techniques and Applications, Lifetime Learning Publications, Belmont, CA, 1985
[15] Storer, James A., Data Compression: Methods and Theory, Computer Science Press, Rockville, MD, 1988
[16] Nelson, Mark, "LZW Data Compression," Dr. Dobb's Journal, Volume 14, Number 10, October 1989, pp 29-37