Available Online at www.ijcsmc.com

International Journal of Computer Science and Mobile Computing

A Monthly Journal of Computer Science and Information Technology

ISSN 2320–088X

IJCSMC, Vol. 3, Issue. 10, October 2014, pg.45 – 51

RESEARCH ARTICLE

An Advanced Text Encryption & Compression System Based on ASCII Values & Arithmetic Encoding to Improve Data Security

Amandeep Singh Sidhu [M.Tech][1], **Er. Meenakshi Garg** [M.Tech][2]
C.S.E. & Guru Kashi University, Talwandi Sabo, Bathinda, Punjab, India
amansidhu0086@gmail.com, +91-99141-85500

Abstract: Compression algorithms reduce the redundancy in data representation thus increasing effective data density. Data compression is a very useful technique that helps in reducing the size of text data and storing the same amount of data in relatively fewer bits resulting in reducing the data storage space, resource usage or transmission capacity. There are a number of techniques that have been used for text data compression which can be categorized as Lossy and Lossless data compression techniques. In this review paper, various lossless data compression techniques have been considered that are in use such as Run Length Encoding, Burrows- wheeler transform, Shannon-Fano coding, Huffman coding, Arithmetic coding and Bit-Reduction algorithm.

Key-Words: Data Compression, Lossless Compression, Huffman Coding, Arithmetic Coding, Run-length Encoding.

1. Introduction:

Data Compression is the process of encoding data so that it takes less storage space or less transmission time that it would if it were not compressed. Compression is possible because most of the real world data is very redundant. Data Compression is a technique that reduces the size of data by removing unnecessary information and duplicity. Data compression is the art of reducing the number of bits needed to store or transmit data.

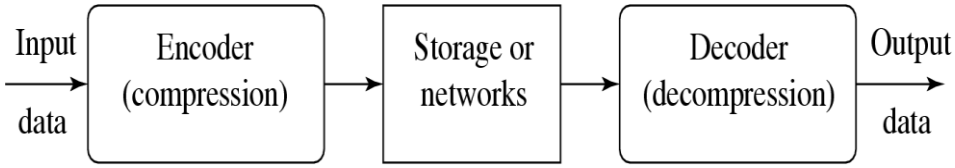

Fig 1: diagrammatic representation of Compression

Mainly, two types of data compression techniques are there: **Lossy and Lossless** data compression. A **lossy data compression method** is one where compressing data and then

decompressing it retrieves data that may well be different from the original, but is "close enough" to be useful in some way. Lossy data compression is used frequently on the Internet and especially in streaming media and telephony applications. These methods are typically referred to as codecs in this context. Most lossy data compression formats suffer from generation loss: repeatedly compressing and decompressing the file will cause it to progressively lose quality. This is in contrast with lossless data compression. **Lossless data compression** makes use of data compression algorithms that allows the exact original data to be reconstructed from the compressed data. This can be contrasted to lossy data compression, which does not allow the exact original data to be reconstructed from the compressed data. Lossless data compression is used in many applications. For example, it is used in the popular ZIP file format and in the Unix tool gzip. It is also often used as a component within lossy data compression technologies. Lossless compression is used when it is important that the original and the decompressed data be identical. Typical examples are executable programs and source code. Some image file formats, notably PNG, use only lossless compression, while others like TIFF and MNG may use either lossless or lossy methods. GIF uses a lossless compression method. Lossless compression methods may be categorized according to the type of data they are designed to compress. Some main types of targets for compression algorithms are text, executables, images, and sound. Whilst, in principle, any general-purpose lossless compression algorithm (general-purpose means that they can handle all binary input) can be used on any type of data, many are unable to achieve significant compression on data that is not of the form that they are designed to deal with. Sound data, for instance, cannot be compressed well with conventional text compression algorithms.

Most lossless compression programs use two different kinds of algorithms: one which generates a statistical model for the input data, and another which maps the input data to bit strings using this model in such a way that "probable" (e.g. frequently Encountered) data will produce shorter output than "improbable" data. The advantage of lossy methods over lossless methods is that in some cases a lossy method can produce a much smaller compressed file than any known lossless method, while still meeting the requirements of the application.

ADVANTAGES OF DATA COMPRESSION

- Less disk space
- Faster writing and reading
- Faster File transfer
- Variable dynamic range
- Byte order independent

DISADVANTAGES OF DATA COMPRESSION

- Effects of errors in transmission
- Added Complication
- Slower for sophisticated methods
- Unknown byte/pixel relationship
- Need to decompress all previous data

2. EXISTING WORK:

Rupinder Singh Brar and Bikramjeet Singh, "A survey on different compression techniques and bit reduction algorithm for compression of text data"
This paper provides a survey of different basic lossless and lossy data compression techniques. On the basis of these techniques a bit reduction algorithm for compression of text data has been proposed by the authors based on number theory system and file differential technique which is a simple compression and decompression technique free from time complexity. This compression algorithm takes O(n) time, where n is the total number of characters in the file and the total computation time required for this algorithm is proportional to O(nlogn). Future work can be done on coding of special character which are not specified on key-board to revise better results.
Agus Dwi Suarjaya, "A new algorithm for data compression optimization"
This paper propose a new lossless data compression algorithm called J-bit encoding(JBE) that manipulates each bit of data inside file to minimize the size without losing any data after decoding. This basic algorithm is intended to be combining with other data compression algorithms to optimize the compression ratio. The performance of this algorithm is measured by comparing combination of different data compression algorithms. The main idea of this algorithm is to split the input data into two data where the first data will contain original nonzero byte and the second data will contain bit value explaining position of nonzero and zero bytes.

Both data then can be compress separately with other data compression algorithm to achieve maximum compression ratio.5 combination algorithms have been tested and compared in this paper. The ability to recognize the hybrid contents (text, audio, video, binary) of a file regardless the file type is potential for future research to achieve better compression ratio.
Ajit Singh and Yogita Bhatnagar, "Enhancement of data compression using Incremental Encoding"
This paper describes the two phase encoding technique which compresses the sorted data more efficiently. The research paper provides a way to enhance the compression technique by merging RLE compression algorithm and incremental compression algorithm. In first phase the data is compressed by RLE algorithm that compresses the frequent occur data bits by short bits. In the second phase incremental compression algorithm stores the prefix of previous symbol from the current symbol and replaces with integer value. This technique can reduce the size of sorted data by 50% using two phase encoding technique.
S. Shanmugasundaram and R. Lourdusamy, "A Comparative Study of Text Compression Algorithms" There are lot of data compression algorithms which are available to compress files of different formats. This paper provides a survey of different basic lossless data compression algorithms. Experimental results and comparisons of the lossless compression algorithms using Statistical compression techniques and Dictionary based compression techniques were performed on text data. Among the statistical coding techniques the algorithms such as Shannon-Fano Coding, Huffman coding, Adaptive Huffman coding, Run Length Encoding and Arithmetic coding are considered. Lempel Ziv scheme which is a dictionary based technique is divided into two families: those derived from LZ77 (LZ77, LZSS, LZH and LZB) and those derived from LZ78 (LZ78, LZW and LZFG). A set of interesting conclusions are derived on their basis. Lossy algorithms achieve better compression effectiveness than lossless algorithms, but lossy compression is limited to audio, images, and video, where some loss is acceptable. The question of the better technique of the

two, "lossless" or "lossy" is pointless as each has its own uses with lossless techniques better in some cases and lossy technique better in others.

U. Khurana and, A.Koul, "Text Compression And Superfast Searching" A new compression technique that uses referencing through two-byte numbers (indices) for the purpose of encoding has been presented. The technique is efficient in providing high compression ratios and faster search through the text. It leaves a good scope for further research for actually incorporating phase 3 of the given algorithm. The same should need extensive study of general sentence formats and scope for maximum compression. Another area of research would be to modify the compression scheme so that searching is even faster. Incorporating indexing so as to achieve the same is yet another challenge.

3. Lossless data compression Techniques:

3.1 Run length Encoding: Run-Length Encoding is a data compression algorithm that is supported by bitmap file formats such as TIFF, BMP. RLE works by reducing the physical size of a repeating string of characters. This repeating string, called a *run*, is typically encoded into two bytes. The first byte represents the number of characters in the run and is called the *run count* and it replaces runs of two or more of the same character with a number which represents the length of the run, followed by the original character; single characters are coded as runs of 1. RLE is useful for highly-redundant data, indexed images with many pixels of the same color in a row, or in combination with other compression techniques also. Here is an example of RLE:

Input: YYYBBCCCCDEEEEEERRRRRRRRR

Output: 3Y2B4C1D6E10R

Although most RLE algorithms cannot achieve the high compression ratios of the more advanced compression methods, RLE is both easy to implement and quick to execute, making it a good alternative for a complex compression algorithm.

3.2 Burrows -Wheeler Transform: Burrows-Wheeler transform does not compress a message, but rather transform it into a form that is more amenable to compression i.e. it can be more efficiently coded by a Run-Length Encoder or other secondary compression technique. The transform rearranges the characters in the input so that that there are lots of clusters with repeated characters, but in such a way that it is still possible to recover the original input. Burrows-wheeler transform (BWT) works in block mode while others mostly work in streaming mode.

The algorithm for a BWT is:

1. Create a string array.
2. Generate all possible rotations of the input string, storing each in the array.
3. Sort the array alphabetically.
4. Return the last column of the array

BWT usually works best on long inputs with many alternating identical characters. Here is an example of the algorithm being run on an ideal input. Note that & is an End of File character:

Input	**Rotations**	**Alpha-sorted rotations**	**Output**
HAHAHA&	HAHAHA&	AHAHA&**H**	**HHH&AAA**
	&HAHAHA	AHA&HA**H**	
	A&HAHAH	A&HAHA**H**	
	HA&HAHA	HAHAHA**&**	
	AHA&HAH	HAHA&H**A**	
	HAHA&HA	HA&HAH**A**	
	AHAHA&H	&HAHAH**A**	

Because of its alternating identical characters, performing the BWT on this input generates an optimal result that another algorithm could further compress, such as RLE which would yield "3H&3A". While this example generated an optimal result, it does not generate optimal results on most real-world data.

3.3 Shannon-Fano coding: This is one of an earliest technique for data compression that was invented by Claude Shannon and Robert Fano in 1949.In this technique, a binary tree is generated that represent the probabilities of each symbol occurring. The symbols are ordered in a way such that the most frequent symbols appear at the top of the tree and the least likely symbols appear at the bottom.

The algorithm to generate Shannon-Fano codes is fairly simple

1. *Parse the input, counting the occurrence of each symbol.*
2. *Determine the probability of each symbol using the symbol count.*
3. *Sort the symbols by probability, with the most probable first.*
4. *Generate leaf nodes for each symbol.*
5. *Divide the list in two while keeping the probability of the left branch roughly equal to those on the right branch.*
6. *Prepend 0 and 1 to the left and right nodes' codes, respectively.*
7. *Recursively apply steps 5 and 6 to the left and right subtrees until each node is a leaf in the tree.*

3.4 Huffman Coding: Huffman coding deals with data compression of ASCII characters. It works in a very similar manner to Shannon-Fano Coding, but the binary tree is built from the top down to generate an optimal result. In Huffman Coding the characters in a data file are converted to binary code. And in this technique the most common characters in the file have the shortest binary codes, and the least common have the longest binary code

The algorithm to generate Huffman codes shares its first steps with Shannon-Fano:

1. *Parse the input, counting the occurrence of each symbol.*
2. *Determine the probability of each symbol using the symbol count.*
3. *Sort the symbols by probability, with the most probable first.*
4. *Generate leaf nodes for each symbol, including P, and add them to a queue.*
5. *While (Nodes in Queue > 1)*

- *Remove the two lowest probability nodes from the queue.*
- *Prepend 0 and 1 to the left and right nodes codes, respectively.*
- *Create a new node with value equal to the sum of the nodes' probability.*
- *Assign the first node to the left branch and the second node to the right branch.*
- *Add the node to the queue*

6. *The last node remaining in the queue is the root of the Huffman tree.*

3.5 Arithmetic Coding: Arithmetic coding is an optimal entropy coding technique as it provides best compression ratio and usually achieves better results than Huffman Coding. It is quite complicated as compared to the other coding techniques. When a string is converted in to arithmetic encoding, the characters having maximum probability of occurrence will be stored with fewer bits and the characters that do not occur so frequently will be stored with more bits, resulting in fewer bits used overall. Arithmetic coding converts the entire input data into a single floating point number.

A general algorithm to compute the arithmetic code is:

1. *Calculate the number of unique symbols in the input. This number represents the base b (e.g. base 2 is binary) of the arithmetic code.*
2. *Assign values from 0 to b to each unique symbol in the order they appear.*
3. *Using the values from step 2, replace the symbols in the input with their codes*
4. *Convert the result from step 3 from base b to a sufficiently long fixed-point binary number to preserve precision.*
5. *Record the length of the input string somewhere in the result as it is needed for decoding.*

Here is an example of an encode operation, given the input "ABCDAABD":

1. There are 4 unique symbols in input, therefore base = 4. Length = 8
2. Values assigned to symbols: A=0, B=1, C=2, D=3
3. Replaced input with codes: "0.01230013_4" where the leading 0 is not a symbol.
4. Convert "0.01231123_4" from base 4 to base 2: "0.01101100000111_2"
5. Result found. Note in result that input length is 8.

Assuming 8-bit characters, the input is 64 bits long, while its arithmetic coding is just 15 bits long resulting in an excellent compression ratio of 24% .

3.6 Bit Reduction algorithm: This algorithm was originally implemented for use in an SMS application. Using this algorithm, it could send about 256 characters per message (typically 160 characters per message) through the same 7-bit GSM network. The idea is, this program reduces the standard 7-bit encoding to some application specific 5-bit encoding system and

then pack into a byte array. This method reduces the size of a string considerably when the string is lengthy and the compression ratio is not affected by the content of the string.

Bit Reduction Algorithm in steps-
1. Pick up a plain text or file which contain the text.

2. Apply Bit Reduction algorithm.

3. Steps to be followed in Bit reduction algorithm-
 Pick up characters from the text file which is to be encoded and obtain their corresponding ASCII code.
 Obtain the binary code from these ASCII key codes corresponding to each character.
 Then put these binary no. into array of byte (8bit array).
 Chop up extra bits from binary no like extra 3 bits from the front.
 Then rearrange these into array of byte and maintain the array.
 Final text will be encoded and as well as compression will be achieved.
 Now decompression will be achieved in reverse order at the client-side.

4. Conclusion:

In this paper, we have reviewed various existing lossless text data compression techniques. For future work A data compression algorithm is to be developed which consumes less time while provides more compression ratio as compared to existing techniques

References

[1] S. Shanmugasundaram and R. Lourdusamy, "A Comparative Study Of Text Compression Algorithms" International Journal of Wisdom Based Computing, Vol. 1 (3), December 2011
[2] Md. Rubaiyat Hasan, "Data Compression using Huffman based LZW Encoding Technique"
[3] V.K. Govindan and B.S. Shajee mohan,"IDBE - An Intelligent Dictionary Based Encoding Algorithm for Text Data Compression for High Speed Data Transmission over Internet "
[4]P.G.Howard and J.C.Vitter, Fellow IEEE, " Arithmetic Coding For Data Compression".
[5] S. kaur and V.S.Verma," Design and Implementation of LZW Data Compression Algorithm", International Journal of Information Sciences and Techniques (IJIST) Vol.2, No.4, July 2012
[6] U .Khurana and,A.Koul, "Text Compression And Superfast Searching", Thapar Institute Of Engineering and Technology, Patiala, Punjab, India-147004
[7] Y. M. Kamir, M. Deris. M. Sufian, and A. A.F. Amri ,"Study of Efficiency and Capability LZW++ Technique in Data Compression", World Academy of Science, Engineering and Technology 35 2009
[8] A.Mukherjee, R.Franceschini, "Data Compression Using Encrypted Text"
[9] D.E.Asuquo, E.O.Nwachukwu and N.J.Ekong, "Application of Run-length Encoding Technique in Data Compression

Regards:
Amandeep Singh Sidhu [M.Tech],
Guide : **Er. Meenakshi Garg** [M.Tech],
C.S.E. & Guru Kashi University, Talwandi Sabo, Bathinda, Punjab, India
amansidhu0086@gmail.com , +91-99141-85500

