AJCSE
ASIAN JOURNAL OF COMPUTER SCIENCE ENGINEERING

www.ajcse.info

Asian Journal of Computer Science Engineering 2017; 2(4):17-21

RESEARCH ARTICLE

Implementation of Lossless Compression Algorithms for Text Data

***B. Usharani**

*Assistant Professor, Department of Computer Science and Engineering, KLEF, Andhra Pradesh, India.

Received on: 05/05/2017, Revised on: 25/07/2017, Accepted on: 10/08/2017

ABSTRACT

Compression is useful to reduce the number of bits needed to represent to data. Compressing saves speed file transfer, and decrease costs for storage hardware and network bandwidth. Compression is applied by a program that uses a algorithm to determine how to shrink the size of data. There are number of data compression techniques used and they can be categorized as Lossy and Lossless Compression methods. This paper examines Huffman and arithmetic lossless data compression algorithms and compares their performance. A set of selected algorithms are implemented to examine the performance in compressing text data. An experimental comparison of a number of two lossless data compression algorithms i.e. arithmetic and Huffman is implemented in this paper. In this paper, some of the Lossless data compression methods and their performance are given.

Keywords: Arithmetic coding, Huffman Algorithm, , Lossless compression, Run length coding, Variable length encoding etc.

INTRODUCTION

Data Compression means to reduce the number of bits to store or to transmit. Data compression techniques are of two types:

1. Lossless data compression-the original data can be reconstructed exactly from the compressed data
2. Lossy data compression--the original data cannot be reconstructed exactly from the compressed data.

With lossless data compression technique every single bit of data is to be remained originally after the decompression .But with lossy compression technique the file is to be reduced permanently by eliminating the redundant data.

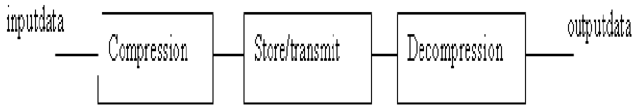


Fig .1 Compression Procedures

To compress the data there is the need to encode the data

The basic coding concepts are

1. Fixed length coding
2. variable length coding

In fixed length coding, same number of bits are used to represent each character. In variable length coding variable number of bits are used to represent a character. To avoid ambiguity no code word is a prefix of another code word. The variable length coding has a uniquely decodable code.

LOSSLESS COMPRESSION TECHNIQUES

ARITHMETIC CODING

Arithmetic coding is based on the frequency of symbols, and the whole message is represented by a fractional number between 0 and 1. As the message size becomes larger, the interval in which the number belongs becomes narrower.

Arithmetic coding Algorithm

The algorithm for encoding a file using this method works conceptually as follows [1]:

1. The "current interval" [L,H) is initialized to [0,1).
2. For each symbol that has to be encoded, do the following procedure:

- Subdivide the current interval into subintervals, one for each possible alphabet symbol. The size of a symbol's subinterval is proportional to the estimated probability that the symbol will be the next symbol in the file according to the model of the input.

***Corresponding Author:** B.Usharani**, Email**: ushareddy.vja@gmail.com

- Select the subinterval corresponding to the symbol that actually occurs next in the file, and make it the new current interval.

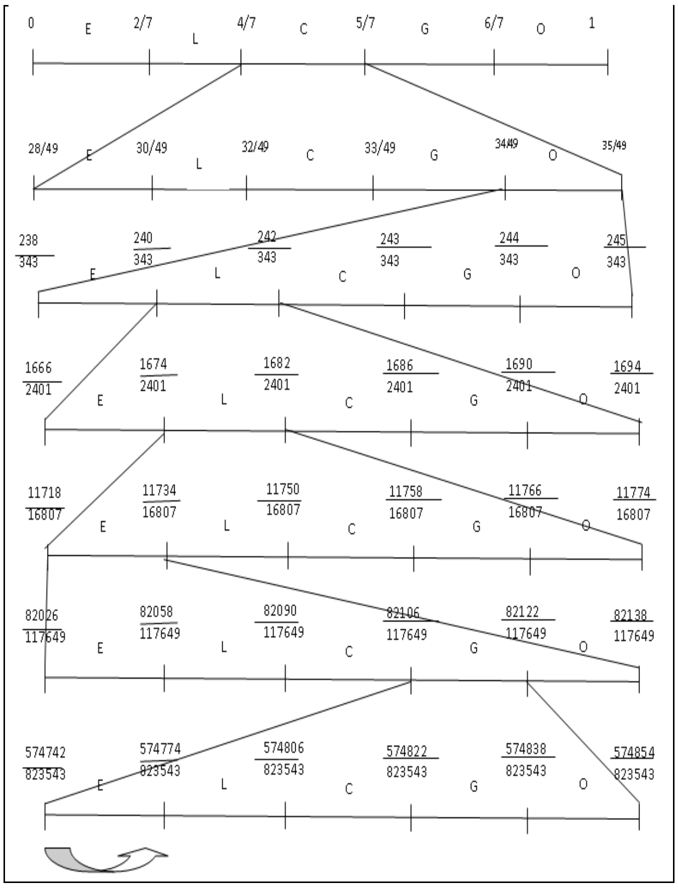
3. Output enough bits to distinguish the final current interval from all other possible final intervals.

Ex: COLLEGE

Table 1.Characters and corresponding frequencies

E	L	C	G	O
2	2	1	1	1

Arithmetic coding completely bypasses the idea of replacing an input symbol with a specific code.
Instead, it takes a stream of input symbols and replaces it with a single floating point number in [0,1).
In arithmetic coding, a unique identifier or tag is generated for the sequence to be encoded.
This tag corresponds to a binary fraction, which becomes the binary code for the sequence.
We conceptually divide the approach into two phases.
In the first phase, a unique identifier or tag is generated for a given sequence of symbols.
This tag is then given a unique binary code
A unique arithmetic code can be generated for a sequence of length *m* without the need for generating code words for all sequences of length *m*.



574760/823543=0.6979113416057206484664431608307

0.10110010101010100101000101010011101101000011110011100111010000...

Fig 2: Implementation of Arithmetic algorithm for the word "COLLEGE"

HUFFMAN ALGORITHM

Huffman constructs a code tree from the bottom up (builds the codes from the right to left).The algorithm starts by building a list of all the alphabet symbols in descending order of their probabilities. It then constructs a tree, with a symbol at every leaf node from the bottom up. This is done in steps, where at each step the two symbols with the smallest probabilities are selected, added to the top the partial tree, deleted from the list, and replaced with an auxiliary symbol representing the two original symbols. When the list is reduced to just one auxiliary symbol, the tree is complete. The tree is then traversed to determine the codes of the symbols [6].

The codes generated using Huffman technique or procedure is called Huffman codes. These codes are prefix codes and are optimum for a given model (set of probabilities). A code in which no codeword is a prefix to another codeword is called a prefix code. The Huffman procedure is based on two observations regarding optimum prefix codes.

1. In an optimum code, symbols that occur more frequently (have a higher probability of occurrence) will have shorter code words than symbols that occur less frequently.
2. In an optimum code, the two symbols that occur least frequently will have the same length. The code words corresponding to the two lowest probability symbols differ only in the last bit. [7]

Though the codes are of different bit lengths, they can be uniquely decoded. Developing codes that vary in length according to the probability of the symbol they are encoding makes data compression possible. And arranging the codes as a binary tree solves the problem of decoding these variable-length codes.[8]
The algorithm for the Huffman coding is

Huffman Algorithm:

1. Create a leaf node for each symbol and add it to the queue.
2. While there is more than one node in the queue:

- Remove the two nodes of highest priority (lowest probability) from the queue
- Create a new internal node with these two nodes as children and with probability equal to the sum of the two nodes' probabilities.
- Add the new node to the queue.

3. The remaining node is the root node and the tree is complete

The technique works by creating a binary tree of nodes. These can be stored in a regular array, the size of which depends on the number of symbols, n. A node can be either a leaf node or an internal node. Initially, all nodes are leaf nodes, which contain the **symbol** itself, the **weight** (frequency of appearance) of the symbol and optionally, a link to a **parent** node which makes it easy to read the code (in reverse) starting from a leaf node. Internal nodes contain symbol **weight**, links to **two child nodes** and the optional link to a **parent** node. As a common convention, bit '0' represents following the left child and bit '1' represents following the right child. A finished tree has up to n leaf nodes and n-1 internal nodes. A Huffman tree that omits unused symbols produces the most optimal code lengths [9].
Huffman coding is a variable length lossless compression coding technique. Huffman compression uses smaller codes for more frequently occurring characters and larger codes for less frequently occurring characters.

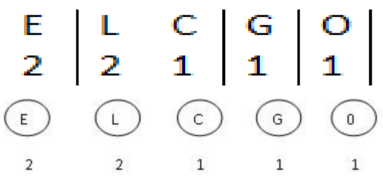


Fig 3: Characters and corresponding frequencies

The characters are arranged according to their frequencies (in descending order).We start by choosing the two smallest nodes There are three nodes with the minimal weight of one. We choose **'G'** and **'O'** and combine them into a new tree whose root is the sum of the weights chosen. We replace those two nodes with the combined tree.

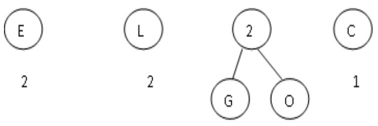


Fig 4: Combining the least two frequencies (G and o) as a single node

Repeat that step, choose the next two minimal nodes, it must be ‘C’ and the combined tree of ‘G’ and ‘O’. The collection of nodes shrinks by one each iteration .we remove two nodes and adds a new one back in.

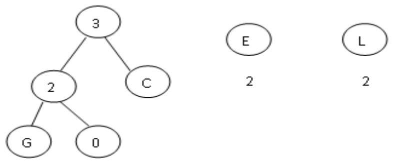


Fig 5: Combining the least two frequencies(c and combined tree of G and o) as a single node

Again, we pull out the two smallest nodes and build a tree of weight 4(i.e. ‘E’ & ‘L’):

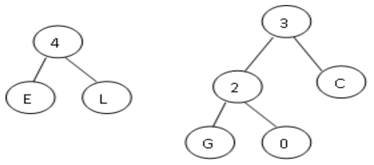


Fig 6: Combining the least two frequencies (E and L) as a single node

Finally, we combine the last two to get the final tree. The root node of the final tree will always have a weight equal to the number of characters in the input file**.**

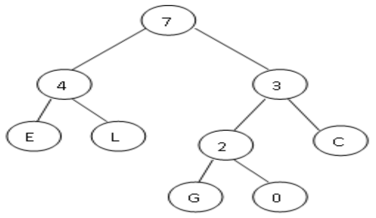


Fig 7: Final Huffman tree

The Huffman tree after assigning 0 to the left child and 1 to the right child will be seen as

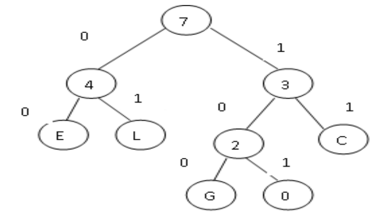
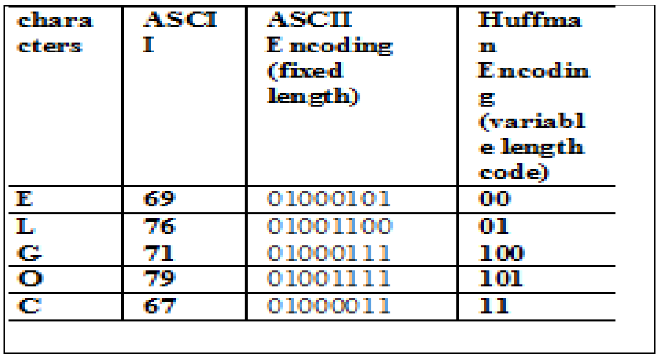


Fig 8 Huffman Tree with 0‘s and 1‘s



characters	ASCII	ASCII Encoding (fixed length)	Huffman Encoding (variable length code)
E	69	01000101	00
L	76	01001100	01
G	71	01000111	100
O	79	01001111	101
C	67	01000011	11

Table 2: Characters and their corresponding ASCII code and variable length code

AJCSE, July-Aug, 2017, Vol. 2, Issue 4

RUN LENGTH CODING

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*. In practice, an encoded run may contain 1 to 128 or 256 characters; the run count usually contains as the number of characters minus one (a value in the range of 0 to 127 or 255). The second byte is the value of the character in the run, which is in the range of 0 to 255, and is called the *run value*.

Uncompressed, a character run of 15 U characters would normally require 15 bytes to store:

UUUUUUUUUUUUUUU

The same string after RLE encoding would require only two bytes:

15U

The 15U code generated to represent the character string is called an *RLE packet*. Here, the first byte, 15, is the run count and contains the number of repetitions. The second byte, U, is the run value and contains the actual repeated value in the run.

In the worst case situations the compressed files are larger than original files. Uncompressed 12 characters are:

abcdefghijkl

The same string after RLE encoding would require24 characters space

a1b1c1d1e1f1g1h1i1j1k1l1

RLE is usually applied to the files that contain large number of consecutive occurrences of the same data.

Run length Coding Algorithm

1. Pick the first character from source string.
2. Append the picked character to the destination string.
3. Count the number of subsequent occurrences of the picked character and append the count to destination string.
4. Pick the next character and repeat steps b) c) and d) if end of string is NOT reached

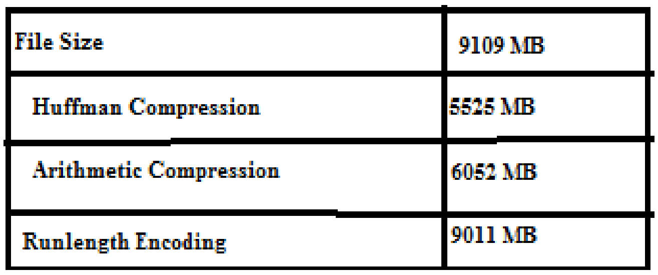
E.g. For the word “COLLEGE “the RLE stored asC1E2G1L2O1.

Run Length Coding Disadvantages

1. Compression ratio is low as compared to other algorithms
2. In the worst case the size of output data can be two times more than the size of input data.
3. Compression efficiency restricted to a particular type of content
4. Mainly utilized for encoding of monochrome graphic data
5. RLE compression is only efficient with files that contain lots of repetitive data

COMPRESSION RESULT

The result after performing the compression over semi-structured data by using the above discussed algorithms is:



File Size	9109 MB
Huffman Compression	5525 MB
Arithmetic Compression	6052 MB
Runlength Encoding	9011 MB

Fig .9: Comparing results

CONCLUSION

In this paper, there is a comparison between Huffman coding and Arithmetic coding and Run length coding techniques of data compression on English words. After testing those algorithms, Huffman coding and Arithmetic coding methodologies are very powerful. Huffman coding and Arithmetic coding gives better results and reduces the size of the text. For small text files RLE is good but for long files Arithmetic gives better results.

REFERENCES

1. https://en.wikipedia.org/wiki/Arithmetic_coding
2. B.Usharani,M.Tanooj Kumar,”Survey on Inverted Index Compression using Semi-structured Data”,By IJIRS, P.No 97-104
3. B.Usharani,”Survey on Inverted Index Compression using Structured Data” by IJARCS ,ISSN No. 0976-5697,P.No(57-61).
4. B.Usharani, M.Tanooj Kumar” Inverted Index Compression using structured Data by AESSN – 2015, ISSN:2348-8387,P.No(119-124).
5. B.Usharani,” Inverted Index Compression over Semi- structured Data By MayFeb journal of Electrical and computer Engineering,vol2(2017),pages 25-31.

AJCSE, July-Aug, 2017, Vol. 2, Issue 4

6. David Solomon —,Data compression, The completeReference‖, Fourth edition, Springer
Khalidsayood,‖ Introduction to data compression‖,Third edition
M.Nelson,J.L.Gailly,‖ The Data CompressionBook‖, second edition
7. http://en.wikipedia.org/wiki/Huffman_coding