International Journal of Information & Computation Technology.
ISSN 0974-2239 Volume 4, Number 9 (2014), pp. 903-910

http://www. irphouse.com

Performance of Lossless Compression Technique

Deepa Raj
Babasaheb Bhimrao Ambedkar University, Lucknow

Seema Gupta
Babasaheb Bhimrao Ambedkar University, Lucknow

Abstract

With the help of compression technique, size of data can be reduced for fast processing and fast saving. Files may be any types such as data files, documents file, images, and so on. In this paper we try to discuss the performance of two widely used lossless compression techniques Huffman coding and Lempel-Ziv-Welsh coding by using encoding technique, decoding technique and compression ratio by taking different size of data files and image files.
Keywords: Compression, Huffman Coding, Lempel-Ziv Coding

1. Introduction

Compression is the process of reducing the total number of bits needed to represent certain information. By this, a smaller file size is generated in order to achieve a faster transmission of electronic files and a smaller space required for its downloading. If the compression and decompression processes induce no information loss, then the compression scheme is lossless; otherwise, it is lossy. **Compression ratio** is the ratio of total byte used for data/image file before compression and total byte used for data/image file after compression.

Compression ratio = B0/B1

*B*0 – number of bits before compression

*B*1 – number of bits after compression

2. Huffman Coding

Huffman coding is a form of statistical coding. Not all characters occur with the same frequency! Yet all characters are allocated the same amount of space Huffman coding is a one kind of coding redundancy technique to compress a file. It changes the fixed length bits to a variable length bits.ASCII code, a fixed-length 7 bit binary code that

encodes 127 characters. Since transmitting data over data links can be expensive. It makes sense to try to minimize the number of bits sent in total. Huffman coding technique takes important role to minimize the number of number of bits. Images, Sound, Video etc. have large data rates that can be reduced by proper coding redundancy techniques. A fixed length code doesn't do a good job of reducing the amount of data sent, since some characters in a message appear more frequently than others, but yet require the same number of bits as a very frequent character.

2.1 Algorithm to make a Huffman tree:

Huffman algorithm for making a huffman tree for variable encoding of each symbols used in the datafile after scanning whole file are as follows :

a.Scan the message to be encoded, and count the frequency of every character.

b. Create a single node tree for each character and its frequency and place into a priority queue, each node has three field frquency field and two address field.

c.(c)Repeat the process Until priority queue contain only one tree.

- Remove the two nodes with the minimum frequencies from the priority queue.
- Create a new node with frequency is sum of two removed frequency and attach these two nodes left side and right side.
- Insert this new node into the priority queue according to the priority.

2.2 Creating a Huffman code:

After development of Huffman tree, code for each symbols used in the text or images can be find by traversing up to the each leaf node, at the time of traversing use the code 0 for leftside traversal and use code 1 for right side traversal, append these code after each traversal then new code is generated for each symbols present in the leaf of the tree.

Suppose we have a 3 character alphabet (ABC) and we want to encode the message ABABAC. Using a fixed length code with 3 bits per character, we need to transmit 6*3=18 bits in total. This is wasteful since we only really need 2 bits to encode the 3 characters ABC. So if we now use a 2 bit code (A=00, B=01, C=10), then we only need to transmit 6*2=12 bits of data. We can reduce this even more if we use a code based upon the frequencies of each character. In our example message, A occurs 3 times, B twice, C once. If we generate a variable length code where A=0, B=10, C=11 , our message above can be transmitted as the bit string: ABABAC
0 10010011= 9 bits

Table 2.2

A	B	C
3	2	1

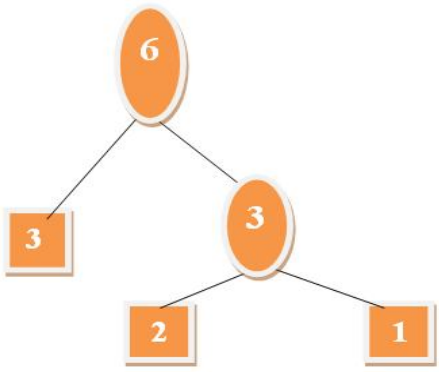

Fig: 2.2 Huffman Tree

3. Lempel-Ziv_Welsh Coding

The Lempel-Ziv-Welsh (LZW) algorithm (also known LZ-78) builds a dictionary of frequently used groups of characters (or 8-bit binary values). Before the file is decoded, the compression dictionary must be sent (if transmitting data) or stored (if data is being stored). This method is good at compressing text files because text files contain ASCII characters (which are stored as 8-bit binary values) but good for graphics files, which may have repeating patterns of binary digits.

3.1 Lempel-Ziv_Welsh coding algorithm:

Algorithm for the compression is explained which shows how new code is generated for the processed string and stored in the dictionary.

1. s = get input character
2. Repeat step 4 while there in no input character
3. c = get input char
4. 4 If s+c is in the dictionary
   a. s = s+c

Else

   a. Output the code for s
   b. Add s+c to the dictionary
   c. S = c
5. 5 End

3.2 Creating a LZW code:

A simple example is to use a 6 character alphabet and a 16 entry dictionary, thus the resulting code word will have 4 bits. Let the transmitted message is:
Then the transmitter and receiver would initially add the following to its dictionary:
0000 ‘a’
0001 ‘b’
0010 ‘c’
0011–1111 empty

First the ‘a’ character is sent with 0000, next the ‘b’ character is sent and the transmitter checks to see that the ‘ab’ sequence has been stored in the dictionary. As it has not, it adds ‘ab’ to the dictionary, to give:

0000 ‘a’
0001 ‘b’
0010 ‘c’
0011 ‘ab’
0100–1111 empty

The receiver will also add this to its table (thus the transmitter and receiver will always have the same tables). Next the transmitter reads the ‘a’ character and checks to see if the ‘ba’ sequence is in the code table. As it is not, it transmits the ‘a’ character as 0000, adds the ‘ba’ sequence to the dictionary, which will now contain:

0000 ‘a’
0001 ‘b’
0010 ‘c’
0011 ‘ab’
0100 ‘ba’
0101–1111 empty

Next the transmitter reads the ‘b’ character and checks to see if the ‘ba’ sequences in the table. As it is, it will transmit the code table address which Identifies it, i.e. 0111. When this is received, the receiver detects that it is in its dictionary and it knows that the addressed sequence is ‘ba’. Next the transmitter reads an ‘c’ and checks for the character in its dictionary. As it is included, it transmits its address, i.e. 0010. When this is received, the receiver checks its dictionary and locates the character ‘c’. This then continues with the transmitter and receiver maintaining identical copies of their

0000 0001 0000 0100 0010
‘a’ ‘b’ ‘a’ ‘ba’ ‘c’
Total bits=20

A great deal of compression occurs when sending a sequence of one character, such as a long sequence of ‘a’. Typically, in a practical implementation of LZW, the dictionary size for LZW starts at 4K (4096). The dictionary then stores bytes from 0 to 255 and the addresses 256 to 4095 are used for strings (which can contain two or more characters). As there are 4096 entries then it is a 12-bit coding scheme (0 to 4096 gives 0 to 212–1 different addresses).

4. Comparative study of Lempel-Ziv-welsh/Huffman coding

This section contains practical examples of programs which use Lempel-Ziv and/or Huffman coding. Most compression programs use either one or both of *Huffman/Lempel-Ziv compression* these techniques. As previously mentioned, both techniques are lossless. In general, **Huffman is the most efficient** but requires two passes over the data, while Lempel-Ziv-welsh uses just one pass. This feature of a single pass is obviously important when saving to a hard disk drive or when encoding and decoding data in real-time communications. Suppose we have a 3 character alphabet (ABC) and we want to encode the message ABABBABCABABBA. Using a fixed length code with 3 bits per character, we need to transmit 14*3=42 bits in total. This is wasteful since we only really need 2 bits to encode the

3 characters ABC. So if we now use a 2 bit code (A=00, B=01, C=10), then we only need to transmit 14*2=28 bits of data. We can reduce this even more if we use a code based upon the frequencies of each character. In our example message, A occurs 6 times, C occurs one times, B occurs 7 times. If we generate a variable length code where A=10, B=0, C=11, our message above can be transmitted as the bit string:

ABABBABCABABBA i.e 10010000100111001000010 total 21 bits. Notice, that as we process the bits each character's code is unique; there is no ambiguity. As soon as we find a character's code, we start the decoding process again.

In this case compression ratio 42/21=2

LZW compression for string "ABABBABCABABBA"

Let's start with a very simple dictionary (also referred to as a "string table"), initially containing only 3 characters, with codes as follows:

code string
---
000 A
001 B
010 C

code string
---
000 A
001 B
010 C

Table 3.2

S	C	Output	code	String
			1	
			2	
			3	
A	B	1	4	AB
B	A	2	5	BA
A	B			
AB	B	4	6	ABB
B	A			
BA	B	5	7	BAB
B	C	2	8	BC
C	A	3	9	CA
A	B			
AB	A	4	10	ABA
A	B			
AB	B			

S	C	Output	code	String
			1	
			2	
			3	
A	B	1	4	AB
B	A	2	5	BA
A	B			
AB	B	4	6	ABB

ABB	A	6	11	
A	EOF	1		ABBA

The output codes are: AB45BC46A. Instead of sending 42 bits, LZW compression scheme sends 27 bits

Compression ratio = 42/27 = 1.56

5. Conclusions and Future scope of the Work

Huffman coding is a technique used to compress files for transmission uses statistical coding. Huffman coding is more frequently used symbols have shorter code words. It Works well for text and fax transmissions and its application that uses several data structures .

Table 5: Compression Ratio Between Huffman and LZW

STRING	HUFFMAN CODING	LZW CODING
ABABBABCABABBA	2	1.56
ABBCCDDAAEEBBFF	1.15	1.04
ABBABBCDABEFAB	1.5	1.27

After comparing we found that Compression ratio of Huffman code is better than LZW. Since LZW take only one pass at the time of encoding and dictionary is also required to send at the time of transition it takes more processing time. Therefore Huffman coding is better for compression of image as well text. When LZW and Huffman are used to compress a binary file (all of its contents either 1 or 0), LZW gives a better compression ratio than Huffman.

References

[1] Compressed Image File Formats: JPEG, PNG, GIF, XBM, BMP, John Miano, August 1999.
[2] Khalid Sayood ,"Introduction to Data Compression", , Ed Fox (Editor), March 2000
[3] Ian H. Witten, Alistair Moffat, Timothy, C. Bel,"Managing Gigabytes: Compressing and Indexing Documents and Images", l , May 1999.
[4] Rafael C. Gonzalez, Richard E. Woods, "Digital Image Processing", November 2001
[5] Data Compression Conference (DCC '00), March 28-30, 2000, Snowbird, Utah
[6] Sayood. Introduction to Data Compression. San Francisco, California, Morgan Kaufmann, 1996.
[7] S. Assche, W. Philips, and I. Lemahieu. Lossless compression of pre-press images using a novel color decorrelation technique. Proc. SPIE Very High Resolution and Quality Imaging III, 3308:85–92, January 1998.
[8] N. Memon, X. Wu, V. Sippy, and G. Miller. An interband coding extension of the new lossless jpeg standard. Proc. SPIE Visual Communications and Image Processing, 3024:47–58, January 1997.
[9] Tzong Jer Chen and Keh-Shih Chuang, A Pseudo Lossless Image Compression Method, IEEE, pp. 610-615, 2010.
[10] Jau-Ji Shen and Hsiu-Chuan Huang, An Adaptive Image Compression Method Based on Vector Quantization, IEEE, pp. 377-381, 2010.
[11] Suresh Yerva, Smita Nair and Krishnan Kutty, Lossless Image Compression based on Data Folding,IEEE, pp. 999-1004, 2011.
[12] Firas A. Jassim and Hind E. Qassim, Five Modulus Method for Image Compression, SIPIJ Vol.3, No.5, pp. 19-28, 2012.