Project Report

on

STRING COMPRESSION THROUGH HUFFMAN ALGORITHM AND LZW(Lempel–Ziv–Welch) ALGORITHM AND THEIR COMPARISON.

submitted towards the partial fulfillment

of the requirement for the award of the

degree of

Bachelor of Technology

in

CS251(DATA STRUCTURES)

submitted by

GIRDHAR JHA(2K20/MC/51)

DEEPAK SINGH(2K20/MC/38)

Subject Teacher:

MS GOONJAN JAIN

ACKNOWLEDGMENT

This project would not have been feasible without the participation and assistance of a large number of people, whose names may not all be listed. Their donations are heartily welcomed and appreciated. However, the team would like to express their gratitude and indebtedness to the following individuals: We are extremely grateful to DTU, particularly Ms. GOONJAN JAIN, our Faculty Advisor, for their guidance and constant supervision, as well as for providing necessary project information and support in completing the project. To all the parents, relatives, friends and others, who in one way or another shared their support, either morally, financially and physically, thank you. Above all, to the Great Almighty, the author of knowledge and wisdom, for His countless love.

GIRDHAR JHA (2K20/MC/51)

DEEPAK SINGH(2K20/MC/38)

INDEX

Abstract

Huffman Coding is a statistical data compression method that reduces the length of the code used to represent the alphabet's symbols. This is a common method for generating Minimum-Redundancy Codes. LZW is a well-known dictionary-based compression algorithm. This means that LZW encodes data by consulting a dictionary rather than tabulating character counts and creating trees. The goal is to start with an initial model, receive data, and update the framework and data encoding, similar to any adaptive and dynamic compression approach. The following question was investigated in this paper: In comparison to each other, whether coding, LZW or Huffman, is more appropriate? The implemented results show that LZW requires no prior information about the input data stream, and also LZW can compress the input stream in one single pass allowing fast execution. LZW coding is more feasible when the high compression ratio and less compression-decompression time are needed.

INTRODUCTION

Data compression is a method of transforming distinct files, such as text, music, and video, into another file so that the original file may be fully retrieved without losing any of the original information. If you need to preserve storage space, this method is critical. Compression is important because it minimises the amount of data that needs to be disseminated and stored. In the reverse of the decompression process, computational resources are widely utilised in the compression process. Data compression is a trade-off between space and time complexity. The level of compression, the amount of distortion, and the processing resources necessary to compress and decompress data are all factors that must be considered while designing data compression techniques. Monochrome images are encoded using the baseline JPEG method utilising Huffman coding [12, 14] and compressed by the provided model and lossless re-encoding procedures to evaluate coding performance. Stuffit is Smith Micro's data compression software, and the most recent version allows you to compress JPEG files.

HUFFMAN CODING

Huffman coding is a complex and lossless compression technique. The characters in a data file are transformed to binary codes, with the shortest binary codes corresponding to the most prevalent characters in the file.

RESEARCH METHODOLOGY

The research methodology generated by the suggested system is described in this section. To determine the performance of the proposed system, we used diverse datasets and conducted various tests.

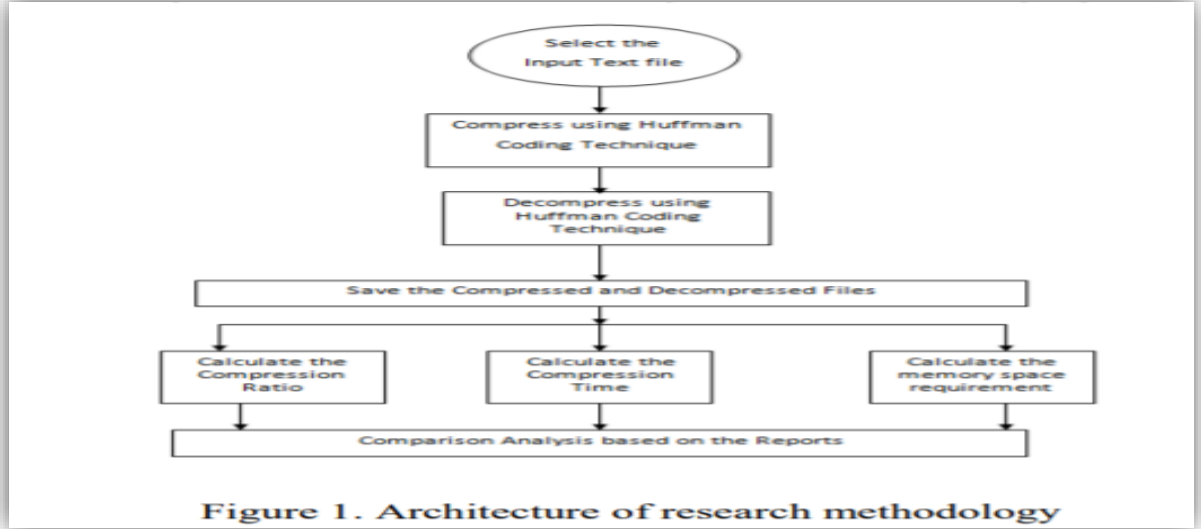


Figure 1. Architecture of research methodology

HUFFMAN CODING TECHNIQUE

"Huffman coding" is a more advanced and efficient lossless compression technique in which characters in a data file are converted to binary codes, with the most common letters having the shortest binary codes and the least common having the longest. The binary codes for the most common characters in the file are the shortest, while the binary codes for the least common characters are the longest. To understand how Huffman coding works, consider a text file that needs to be compressed and the characters in the file have the following frequencies:

Table 1. Huffman Coding Table

Symbols	frequencies
A	8
B	10
E	32
G	12
K	9
M	66

Step 1: Arranging in the ascending order A:8 K:9 B:10 G:12 E:32 M:66

Step 2: making tree of lowest two 17 B:10 G:12 E:32 M:66

A:8 K:9

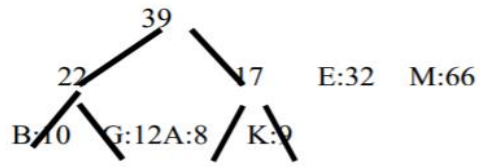
Step 3: Arranging in the ascending order B:10 G:12 17 E:32 M:66

A:8 K:9

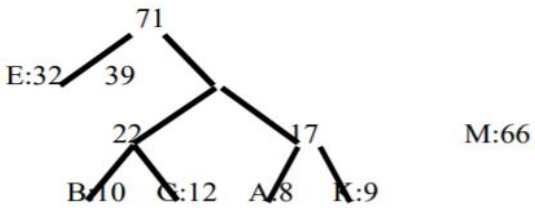
Step 4: making tree of lowest two 22 17 E:32 M:66

B:10 G:12 A:8 K:9

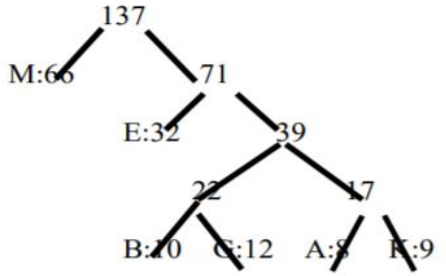
Step 5: making tree of lowest two



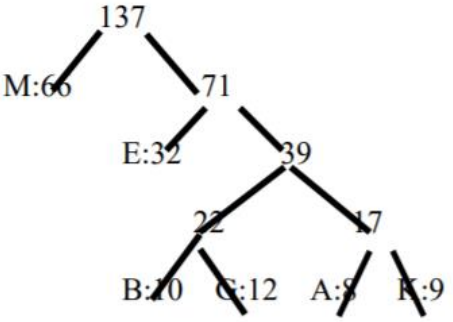
Step 6: making tree of lowest two



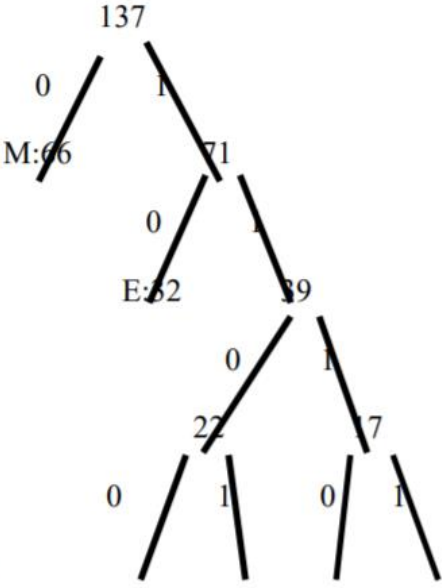
Step 7: making tree of lowest two



Step 7: making tree of lowest two



Step 8: Assigning bit to the tree



B:10 G:12 A:8 K:9

Step 9: Fill bit to the table

Table 2. Huffman Coding Table

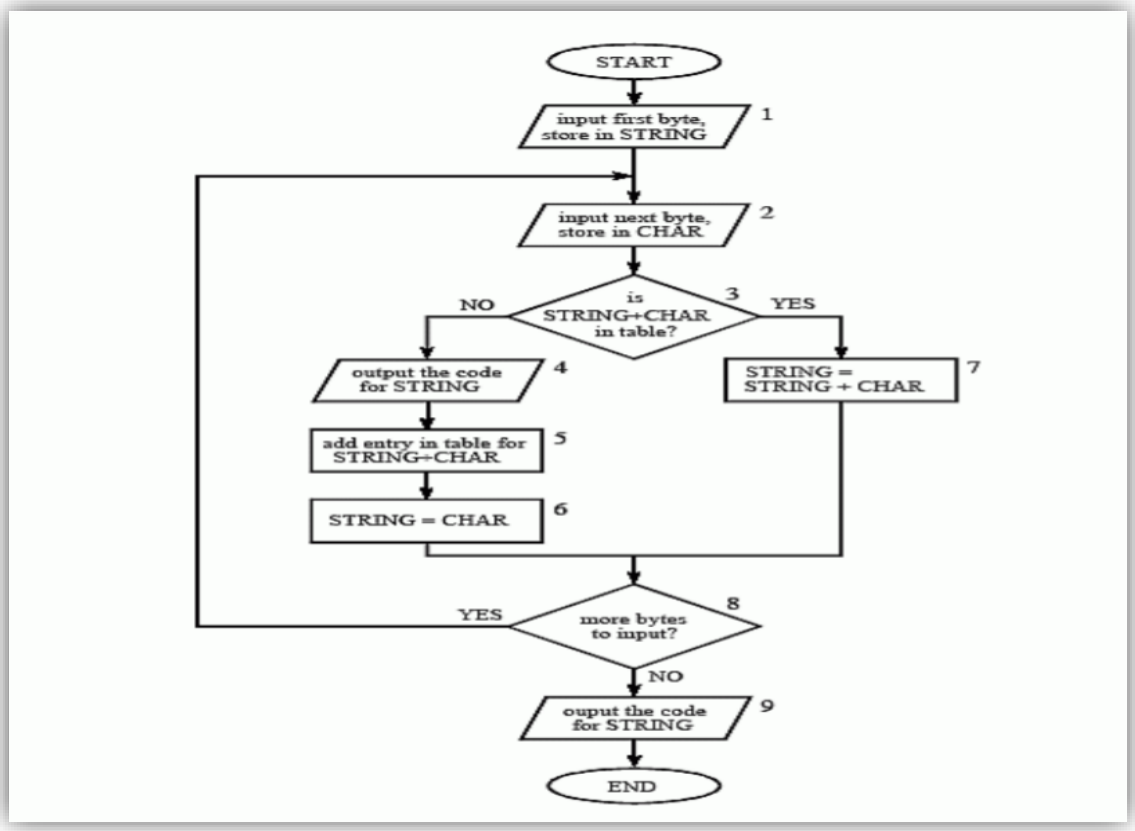
Symbols	frequencies	Bit
A	8	1110
B	10	1100
E	32	10
G	12	1101
K	9	1111
M	66	0

LZW DATA COMPRESSION

A dictionary maintains communication between the longest words and codes when the input data is processed. The words are replaced with subsequent codes, compressing the input file. As the number of big words grows, the algorithm's effectiveness grows.

ENCODING

The single-character strings matching to all potential input characters (and nothing else save the clear and stop codes if they're utilised) are initialised in a dictionary. The method operates by looking for longer substrings in the input string until it discovers one that isn't in the dictionary. When such a string is discovered, the index for the string less the last character (i.e., the dictionary's longest substring) is retrieved and delivered to output, while the new string (containing the last character) is added to the dictionary with the next available code.



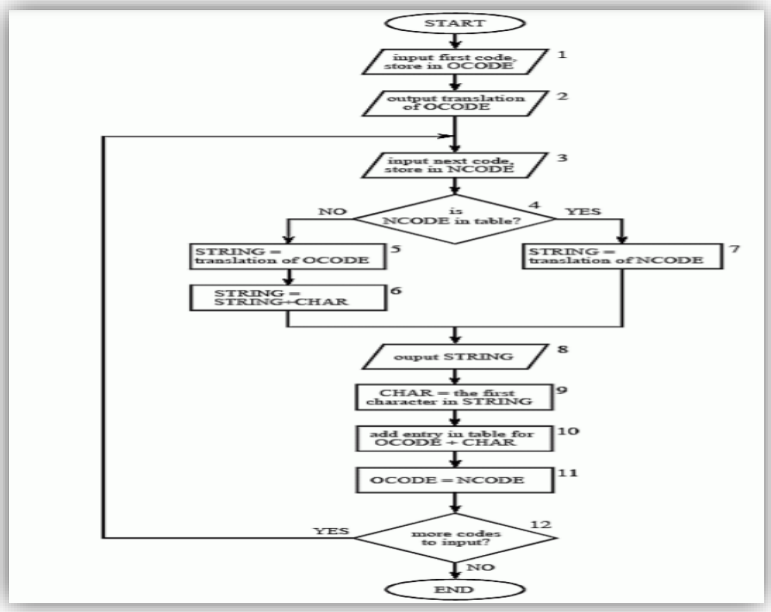
EXAMPLE FOR AN ENCODING PROCESS

Input String = /WED/WE/WEE/WEB/WET			
Character Input	Code Output	New code value	New String
/W	/	256	/W
E	W	257	WE
D	E	258	ED
/	D	259	D/
WE	256	260	/WE
/	E	261	E/
WEE	260	262	/WEE
/W	261	263	E/W
EB	257	264	WEB
/	B	265	B/
WET	260	266	/WET
EOF	T		

The Compression Process: The sample output for the string is shown in above chart along with the resulting string table. As can be seen, the string table fills up rapidly, since a new string is added to the table each time a code is output.

DECODING

Reading a value from the encoded input and outputting the equivalent string from the initialised dictionary is how the decoding process works. Simultaneously, it gets the next value from the input and adds the concatenation of the string just output and the first character of the string obtained by decoding the next input value to the dictionary. The decoder then moves on to the next input value (which was already read in as the "next value" in the previous pass) and repeats the process until no more input is available, at which time the last input value is decoded without further dictionary.



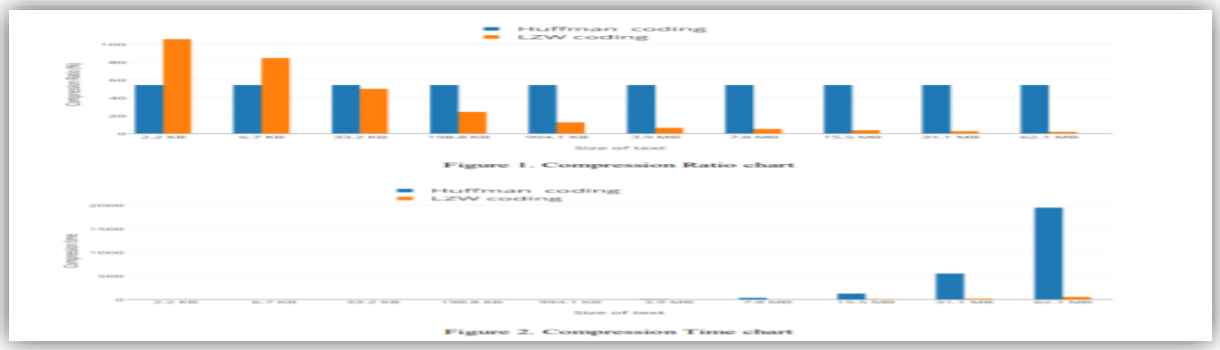
DECOMPRESSION FLOWCHART

Input Codes: / W E D 256 E 260 261 257 B 260 T				
Input/ NEW_CODE	OLD_CODE	STRING/ Output	CHARACTER	New table entry
/	/	/		
W	/	W	W	256 = /W
E	W	E	E	257 = WE
D	E	D	D	258 = ED
256	D	/W	/	259 = D/
E	256	E	E	260 = /WE
260	E	/WE	/	261 = E/
261	260	E/	E	262 = /WEE
257	261	WE	W	263 = E/W
B	257	B	B	264 = WEB
260	B	/WE	/	265 = B/
T	260	T	T	266 = /WET

Compression Ratio: The compression ratio describes the difference in file size between an uncompressed image and a compressed version of the same image.

Compression Ratio = Size of the original image /Size of the compressed image.

RESULTS Table 3 summarises the results of the techniques, including Huffman and LZW compression and decompression times. Parameters of performance: Compression Ratio, Compression Time, and Decompression Time are used to evaluate the compression algorithm's performance. • Compression ratio: The difference between the size of the input text and the compressed text. (C2/C1) *100% is the compression ratio. • Total Time to Run the Compression Method: The total time it takes to run the compression algorithm. • Decompression Time: The time it takes to run the decompression algorithm in its entirety. C1 refers to the size of the input text before compression. C2 denotes the size of the text after compression.



	Huffman Coding			Lempel–Ziv–Welch (LZW) Coding		
Input text size	**Compression Ratio(%)**	**Compression Time**	**Decompression Time**	**Compression Ratio (%)**	**Compression Time**	**Decompression Time**
2.2 KB	54.20	0.0	0.0156	105.56	0.0	0.0
6.7 KB	54.20	0.0	0.069	84.46	0.0	0.0
33.2 KB	54.205	0.015	0.231	49.94	0.0156	0.0156
198.8 KB	54.204	0.0625	1.402	24.219	0.052	0.0159
994.1 KB	54.203	0.601	7.076	12.458	0.284	0.115
3.9 MB	54.203	8.574	30.029	6.319	1.622	1.624
7.8 MB	54.203	34.266	55.879	5.062	3.224	6.852
15.5 MB	54.203	123.664	115.858	3.604	8.560	29.734
31.1 MB	54.203	548.473	237.528	2.566	17.352	103.317
62.1 MB	54.203	1945.389	470.933	1.827	53.198	469.634

o

REFERENCE

[1] https://en.wikipedia.org/wiki/Huffman_coding

[2] http://www.geeks3.forgeeks.org/lzw-lempel-ziv-welchcompression-technique/

[3] https://users.cs.cf.ac.uk/Dave.Marshall/Multimedia/node210.ht ml

[4] https://www.researchgate.net/publication/220114874_A_Memo ry-Efficient_and_Fast_Huffman_Decoding_Algorithm

[5] https://www.researchgate.net/publication/4140913_A_Memory efficient_Huffman_Decoding_Algorithm

[6] http://www/acm/org/crossroads/xrds6-3/sahaimgcoding.html.

[7] “Introduction to Data Compression”, by Guy E. Blelloch, Carnegie Mellon University, (October 16, 2001).

[8] M. Rabbani, and P. W. Jones, “Digital Image Compression Techniques”, (Edition-4), p. 51, 1991. [8] S. Saha, “Image Compression – from DCT to Wavelets: A Review”, Unpublished.

[9] Blelloch, G.: 'Introduction to Data Compression'. Carnegie Mellon University, September , 2010.

[10] Jian-Jiun Ding and Tzu-Heng Lee, "ShapeAdaptive Image Compression", Master’s Thesis, National Taiwan University, Taipei, 2008