International Journal of Engineering Applied Sciences and Technology, 2017
Vol. 2, Issue 3, ISSN No. 2455-2143, Pages 38-44
Published Online January-February 2017 in IJEAST (http://www.ijeast.com)

A HUFFMAN ALGORITHM OPTIMAL TREE APPROACH FOR DATA COMPRESSION AND DECOMPRESSION

Nirmal Sharma
Research Scholar, Computer Science
Teerthanker Mahaveer University Moradabad,
U.P., INDIA

S. K. Gupta
Computer Science. & Engineering.
Department of BIET, Jhansi,
U.P., INDIA

Abstract- **Data compression is very successful approach for storing, retrieving, and transmitting data at the optimum resources. As the uses of databases are dramatically increasing, the data compression paradigm to reduce the size of data warehouse contents require as basic infrastructure for the organization. The objective of this research work is to reduce the size of data warehouse contents during execution time. The Information Technology has a large number of implementation data compression algorithms that compresses the data and reduces the size of database. It is tough task to reduce the size of contents of database and minimize the time consumption. In this paper we present steps by step algorithm to minimize the size of data warehouse by compressing the database. The algorithm is very simple, clear and specific; this algorithm of data compression and decompression in data warehouse of tree based structure. The algorithm will operates each bit of data exclusive file to reduce the size without losing any data afterward translating which is classified to lossless data compression. This approach is quantified and the business user can operate the organization report using this algorithm.**

Keywords: *Compression Algorithm, data compression, data decompression, data warehouse, optimal tree, buffer overrun.*

I. INTRODUCTION

Almost every organization needs database for keeping information and their entire decision making system depends upon the proper functioning of databases. As the competition and pressure increases it's become a primary objective of all the organization to store data in such a way that if any one requires the data it can be transmitted through network easily and retrieve the data accordingly. The large volume of data sometimes leads a buffer overrun problem. A buffer overflow condition occurs when a program attempts to read or write outside the bounds of a block of allocated memory or when a program attempts to put data in a memory area past a buffer [1][2]. The data compression paradigm is the best way to compact the data that require minimum bits space as compare to their original representation. The structure and function of data warehousing and data mining in any organization plays a vital role for the smooth functioning and their success. Large volume or complex data always creates problem during storing, transmitting and retrieving. Therefore, data compression is desperately required by all the organization for smooth, reliable and timely availability of the information. There is a number of data compression algorithm developed by researchers to compress the data and image files. A little work has been done in this research paper to reduce the size of the data and minimize the access time of fetching the records from the data warehouse [3].
Huffman data encoding paradigm based on the lossless compression of frequency of data occurrence in a file that is being compressed. The Huffman algorithm is statistical coding based technique, which means the more probable the occurrence of a symbol is; the shorter will be its bit-size representation. In any file, some characters are used more than others characters. Using binary representation, the number of bits required to represent each character depends upon the number of characters that have to be represented. Using one bit we can represent two characters, i.e., 0 represents the first character and 1 represents the second character. Using two bits we can represent four characters, and so on.
The amount of space required to store each ASCII character. Uncommon characters need important treatment; they require the same 8 bits that are used for much rarer characters. A file whose size is larger

characters encoded using the ASCII scheme will take largest bytes size. A fixed-length encoding like ASCII is convenient because the boundaries between characters are easily determined and the pattern used for each character is completely fixed (i.e. 'A' is always exactly 65). This approach is using optimal tree and balanced tree encoding and decoding data [4].

This paper organized as the general introduction of data compression and designs the balanced tree by Huffman algorithm in section-1. The algorithm related to data compression and decompression proposed in section-2, followed by the results obtained using Huffman algorithm in section-3, finally the future work and conclusion are compiled in section-4, references are mention in section-5.

II. PROPOSED ALGORITHM

The proposed algorithm works on tree nodes of data that should be reduced in size and enhance the input mechanism handle by Huffman coding. This algorithm is to divide into two parts first is encoding and second one is decoding data with tree base structure.

2.1 Using: Huffman Coding Algorithm

The ASCII set has 256 characters with identical occurrence. In this table has used 90 and unique characters. This algorithm precedes assistance of the difference between occurrences and practices little bit space for the frequently occurring characters at the amount of consuming to use more space for each of the more infrequent characters. For instance as tree structure encoding form—it is involved bits format. The reserves character from not consuming to practice a full 8 bits for the most corporate characters makes up for consuming to practice more than 8 bits for the infrequent characters and the overall result is that the file virtually always desires little bit space [5].

2.1.1 A Tree Structure as encoding form

Binary tree show each element for encoded form and exact figure. Each character is kept at a leaf node. Any exact character encoding is found by outlining the way from the root to its node. Each left-going edge denotes a 0, each right-going edge a 1 as shown in figure-1. For instance, this tree statistics the compressed fixed-length encoding we recognized earlier:

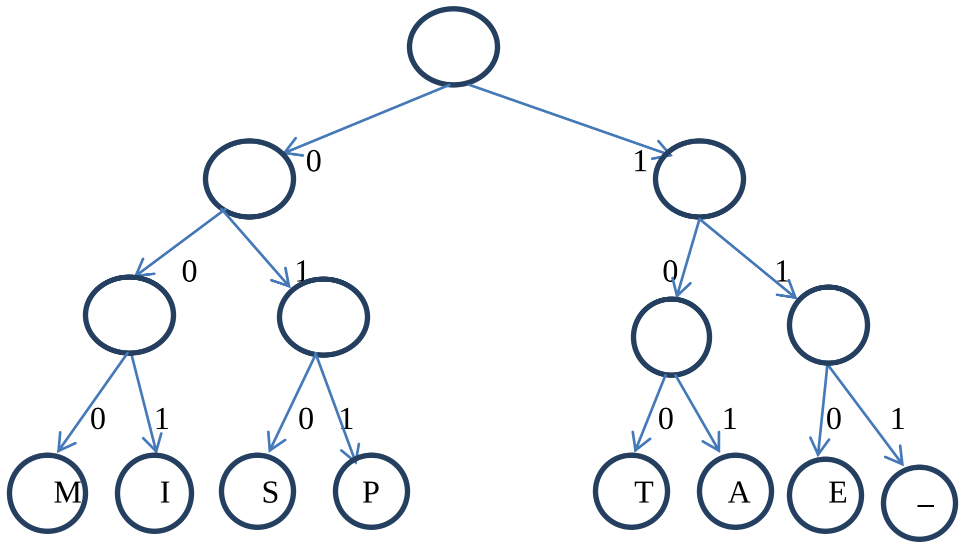
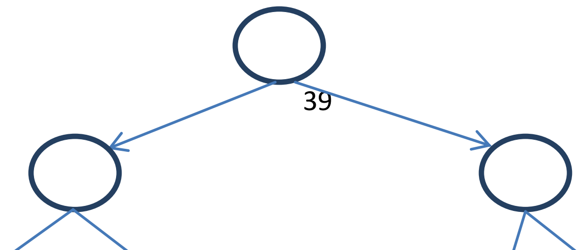


Figure 1 Tree structure of Huffman coding

Overhead tree shows, the encoding for **'P'** can be resolute through outlining the way from the root to the **'P'** node. Successful left before right then right over denotes a **011** encoding.

Nowadays, in this figure shows as a tree for the variable-length Huffman encoding we were consuming



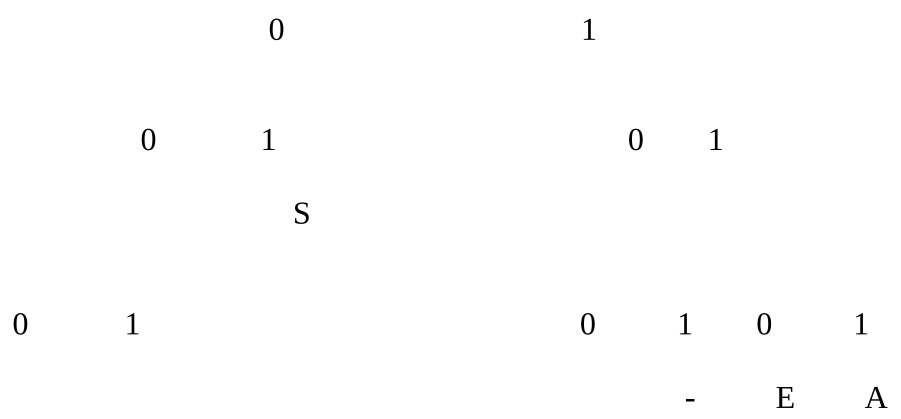


Figure 2 Second Level Tree structure of Huffman coding

The way to **'S'** is impartial left right or **01**; the way to **'A'** is right-right-right-right or **111**. The prefix belongs of the Huffman encoding is visually denoted by the element that characters only conquer leaf nodes as shown in figure-2.
The tree presented above for **"MISSISSIPPI STATE"** is, in fact, an optimal tree—there are no other tree encodings by character that use fewer than 42 bits to encode this string. There are other trees that use exactly 42 bits; for example you can simply exchange any sibling nodes in the above tree and get a different but equally optimal encoding.

The Huffman tree doesn't seem as balanced as the fixed-length encoding tree. You have received in our argument on binary search trees that an unbalanced tree is bad thing. However, when a tree represents a character encoding, that imbalance is actually a good thing. The shorter routes represent those often occurring characters that are being encoded with fewer bits and the longer routes are used for more rare characters. Our plan is to shrink the total number of bits required by shortening the encoding for some characters at the expense of lengthening others. If all characters occurred with equal frequency, we would have a balanced tree where all routes were roughly equal. In such a situation we cannot attain much compression since there are no real replications or patterns to be broken [6].

2.1.2 A Tree Structure using Decoding form

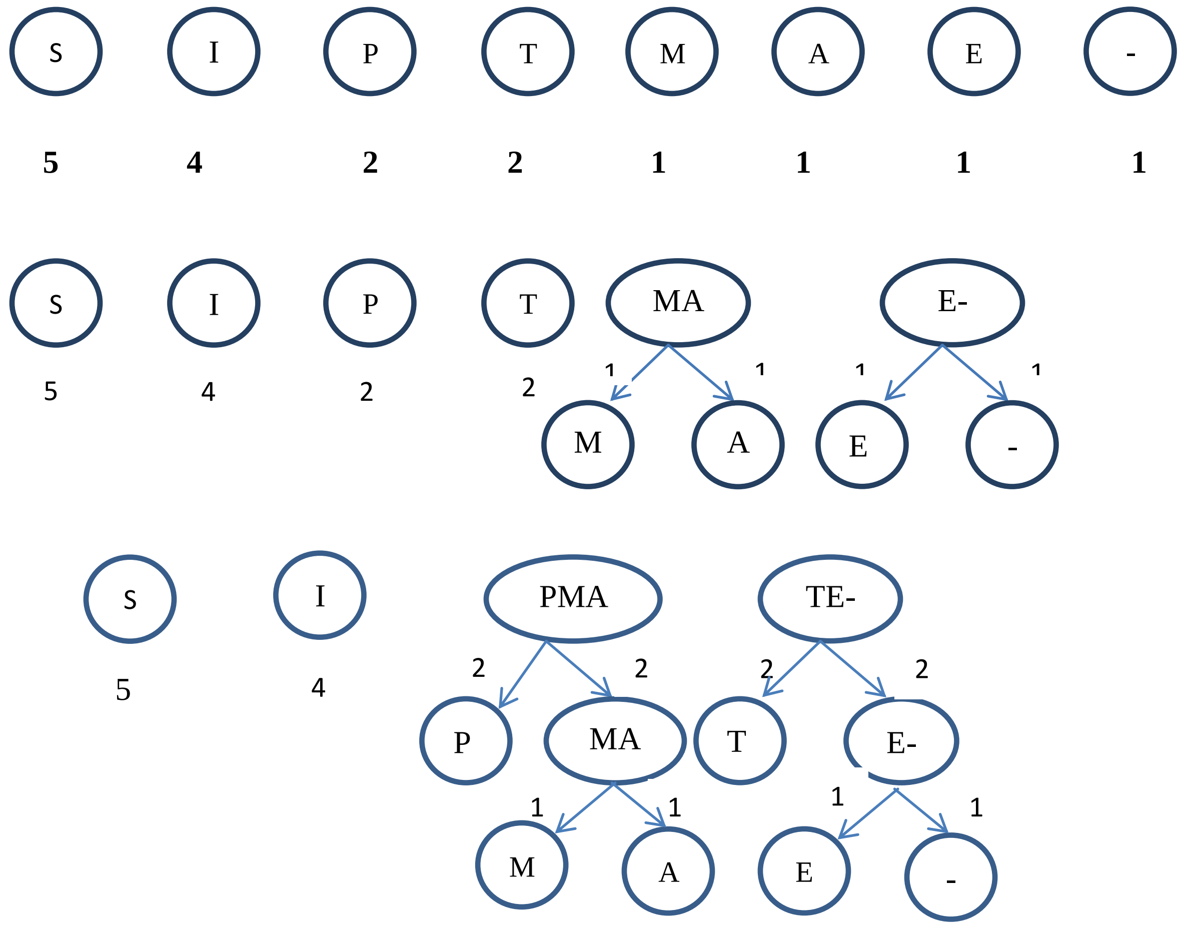
In this figure shows encoding such as a tree easily converted into decoding form. We are used fixed-length tree to decode the stream of bits **0111000100100010**. Twitch at the creation of the bits and at the origin of the tree. The original bit is **0**, thus suggestion one stage to the left, the succeeding bit is **1**, thus monitor right from here, and we have present powerful at a leaf, which displays that we have impartial finalized assessment the bit design for a distinct character. Detecting at the leaf's make, in this research paper, we impartial input a **'S'**. Present we prefer wherever we left off in the bits and twitch finding over from the origin. Finding **110** leads us to **'E'**. Current over the outstanding bits and we decode the string **"SETTMI"**.

Producing an optimal tree
Question is raised here, how did we create distinct tree? We necessary an algorithm for producing the optimal tree giving a minimal per-character encoding for a exact file. The algorithm we will practice here was developed by D. Huffman in 1952.

The Huffman tree is producing the character; individually character becomes a load identical to the number of times it happens in the file. For instance, in the **"MISSISSIPPI STATE"** the character **'S'** has load 5, **'I'** has load 4, the 'P' and 'T' have load 2, and the other characters have load 1. Our initial job is to figure these loads, which we can prepare by a simple permit over the file to become the occurrence computations. For each character, we generate free tree node covering the character assessment and its consistent load. A tree has each node through impartial one access. The main purpose to association, all these distinct trees are an optimal tree through connecting them organized from the lowest upwards [7].

The common method following steps are:

1. Singleton trees are created a collection of each character, each character load identical to the character frequency as shown in figure-3.
2. Since the gathering, pick out the two trees with the lowest loads and eliminate them. Combine them into an original tree whose root has a load identical to the amount of the loads of the two trees and with the two trees as its left and right sub trees.
3. Increase the original collective tree back into the gathering.
4. Replication steps 2 and 3 until there is only one tree left.
5. The outstanding node is the origin of the optimal encoding tree.



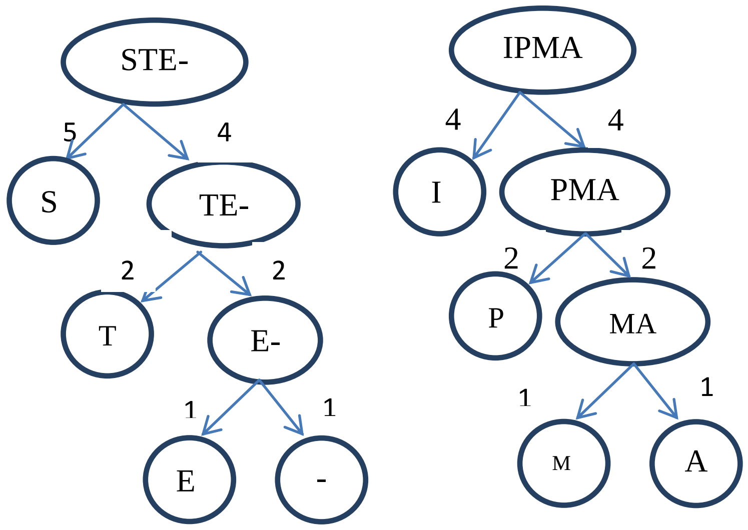


Figure 3 Step-1 to 4 Tree structure of Huffman Decoding

Huffman coding has needs information of occurrences for each representation from alphabet. The output of compressed by the Huffman tree through the Huffman codes for signs or just the occurrences of signs which are applied to make the Huffman tree should be deposited. This data is desired throughout the decoding process and it is located in the caption of the compressed file. The algorithm produces codes that are supplementary active than static Huffman coding. Loading Huffman tree beside by the Huffman codes for signs with the Huffman tree is not desired here [8][9].

III. RESULT OF DATABASE

Table 1 Access Time Performance of Database

Database (n)	**Size_of_Database**	**Access Time(Milliseconds) (X)**
NIU_CS	110040	2250
NIU_EC	110730	2280
NIU_CE	120060	2303
NIU_ME	198310	2859
NIU_BT	200830	2908
NIU_EE	200940	2950
NIU_EEE	210010	2985
NIU_SBM	220050	2998
NIU_SLA	250000	3031
NIU_SOS	260000	3020
NIU_LAW	280000	3060
NIU_NURSING	290000	3090
NIU_ARCH	300000	3100

NIU_EDU	305000	3150
NIU_FINEART	310000	3190
NIU_MEDIA	310500	3193
NIU_BDS	320500	3200
NIU_MBBS	330000	3290
NIU_BSW	330500	3300
NIU_MSW	**340000**	**3350**

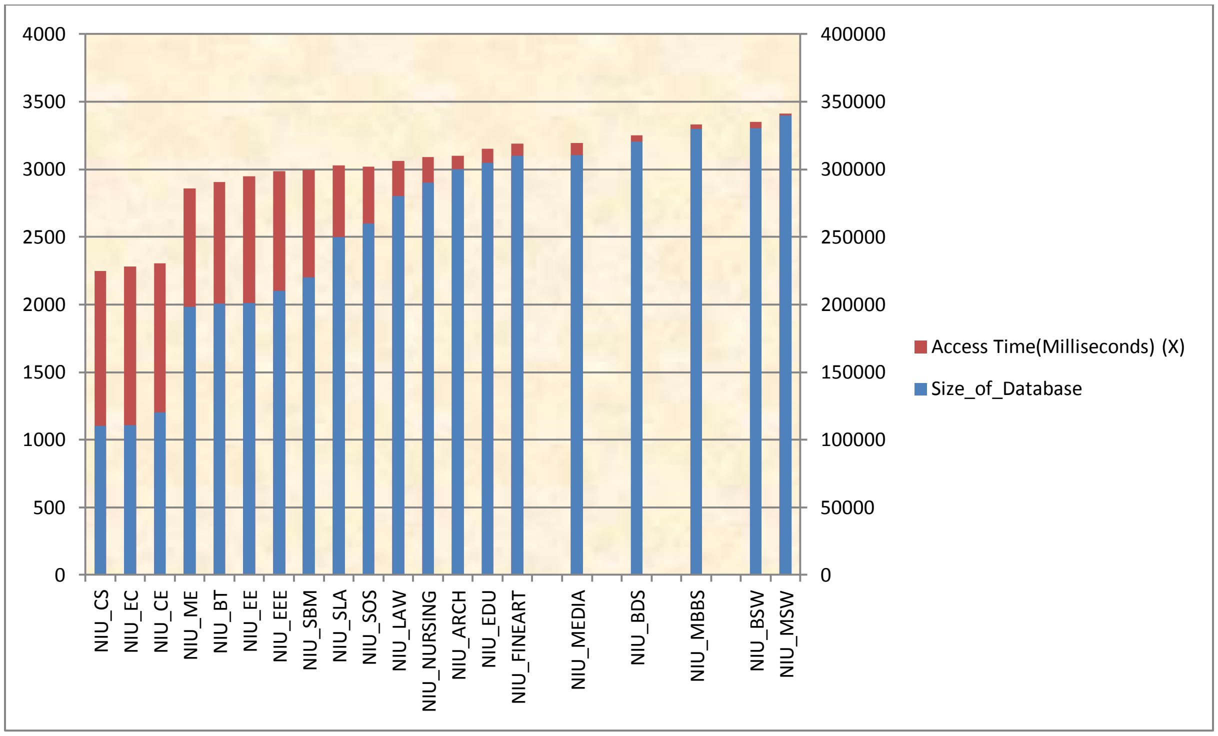


Figure 4 Access Time Performance of Database

A multiple database can use above algorithm so that the organization has a large volume of data which has to be maintained by using a particular algorithm. A database stores large amounts of departmental information and all details of a department. The above information can be used to solve a number of queries, and it can be used to increase the functionality of an organization in a short time. Not only users are able to access the database contents, but this data is also providing integrity [10][11]. Not only they must be able to do this work, but also able to manage this information with current data. When it manages database, we can do more functions on this database other than exchange. All the above types of problem can be solved by proposed algorithm. The figure-4 depicts that as the size of database increases the performance related to access time is decreasing subsequently.

IV. CONCLUSION

In this paper shows data compression and decompression algorithm using Huffman coding approach with balanced tree based structure. By research consuming different categories of files with 20 records of each category was showed result. This algorithm stretches improved compression ratio time is compressed. For the purpose that roughly files involve of hybrid contents as audio video and text etc. the capability to identify substances irrespective the file category, divided it then compresses it distinctly with suitable algorithm to the substances is possible for supplementary research in the upcoming to attain improved compression ratio. We observed at the structure of database and the numerous phases of compression and the decompression, in which the phases are track in opposite direction compared to the compression. HEC Using Huffman coding, we can translate the communication into a string of bits and send it to you. However, you cannot decompress the communication, because you don't have the encoding tree that we used to direct the communication. Huffman coding has two features: It needs only one pass over the input and it improves tiny or no overhead to the output. This algorithm has to restructure the complete Huffman tree after encoding each sign which converts slower than the Huffman coding.

V. REFERENCES

[1] Mahtab Alam, Prashant Johri, Ritesh Rastogi, Buffer Overrun: Techniques of Attack and Its Prevention, International Journal of Computer Science & Emerging Technologies (E-ISSN: 2044-6004), Volume 1, Issue 3, October 2010, pp: 1-6.

[2] Jonathan Pincus, Brandon Baker, Beyond Stack Smashing: Recent Advances in Exploiting Buffer Overruns", IEEE Coputer Society, 2004, pp.20-27.

[3] Capo-chichi, E. P., Guyennet, H. and Friedt, J. K-RLE a New Data Compression Algorithm for Wireless Sensor Network. In Proceedings of the 2009 Third International Conference on Sensor Technologies and Applications.

[4] I Made Agus Dwi Suarjaya A New Algorithm for Data Compression Optimization (IJACSA) International Journal of Advanced Computer Science and Applications, Vol. 3, No.8, 2012

[5] Bell, T.C., Witten, I.H., Cleary, J.G.: Calgary Corpus: .Modeling for text compression., Computing Surveys 21(4), 557-591, 1989.

[6] Campos, A. S. E. Move to Front. Available: http://www.arturocampos.com/ac_mtf.html (last accessed July 2012).

[7] Julie Zelenski, Keith Schwarz,"Huffman Encoding and Data Compression", Spring May 2012

[8] Gallager R.G., "Variations on a theme by Huffman", IEEE Transactions on Information Theory, Vol. 24, No. 6, pp. 668-674, 1978.

[9] Huffman D.A., "A method for the construction of minimum-redundancy codes", Proceedings of the Institute of Radio Engineers, Vol. 40, No.9, pp. 1098–1101, 1952.

[10] Vitter J.S., "Design and analysis of dynamic Huffman codes", Journal of the ACM, Vol. 34, No. 4, pp.825-845, 1987.

[11] Vitter J.S., "Dynamic Huffman coding", ACM Transactions on Mathematical Software, Vol. 15, No.2, pp. 158-167, 1989.