Volume 6, No. 3, May 2015 (Special Issue)
ISSN No. 0976-5697
International Journal of Advanced Research in Computer Science
RESEARCH PAPER
Available Online at www.ijarcs.info

Survey on Inverted Index Compression over Structured Data

B.Usharani
Dept.of Computer Science and Engineering
Andhra Loyola Institute of Engineering and Technology
India
e-mail:usharani.bh@gmail.com

M.TanoojKumar
Dept.of ComputerScience and Engineering
Andhra Loyola Institute of Engineering and Technology
India
e-mail:mtanooj@gmail.com

Abstract: A user can retrieve the information by providing a few keywords in the search engine. In the keyword search engines, the query is specified in the textual form. The keyword search allows casual users to access database information. In keyword search, the system has to provide a search over billions of documents stored on millions of computers. The index stores summary of information and guides the user to search for more detailed information. The major concept in the information retrieval(IR) is the inverted index. Inverted index is one of the design factors of the index data structures. Inverted index is used to access the documents according to the keyword search. Inverted index is normally larger in size ,many compression techniques have been proposed to minimize the storage space for inverted index. In this paper we propose the Huffman coding technique to compress the inverted index. Experiments on the performance of inverted index compression using Huffman coding proves that this technique requires minimum storage space as well as increases the key word search performance and reduces the time to evaluate the query.

Keywords: Huffman Algorithm, inverted index, index compression ,keyword search, Lossless compression, Structured data, Variable length encoding.

I. INTRODUCTION

Keyword search is the mechanism used for information discovery and retrieval. Keyword search is a simple search model that can be issued by writing a list of keywords. Keyword based search enables users to easily access databases without the need either to learn a structured query language or to study complex data schemas. The essential task in keyword search is to display query results which automatically gather relevant information that is generally fragmented and scattered across multiple places.

Search engine index size is hundreds of millions of web pages. Search engines have to answer tens of millions of queries every day. The purpose of storing an index is to optimize speed and performance in finding relevant documents for a search query. Without an index, the search engine would scan every document in the corpus, which would require considerable time and computing power. For example, while an index of 10,000 documents can be queried within milliseconds, a sequential scan of every word in 10,000 large documents could take hours. The additional computer storage required to store the index, as well as the considerable increase in the time required for an update to take place, are traded off for the time saved during information retrieval.[1]

The first Google Index in 1998 already has 26 million pages, and by 2000 the Google index reached the one billion mark.[2]

The inverted index data structure is a key component of keyword search. The index for each term can be sorted in order of allowing ranked retrieval documents. Search engines use the traditional docid ordering, where each posting list is ordered by ascending document id, which permits efficient retrieval. The compression of the docid ordering of posting lists is the focus of our work. The proposed system deals with the document id(docid) and term frequency (tf) compression.

A. Structured Data

Data that resides in a fixed field within a record or file is called structured data. This includes data contained in relational databases and spreadsheets. Structured data first depends on creating a data model – a model of the types of business data that will be recorded and how they will be stored, processed and accessed. This includes defining what fields of data will be stored and how that data will be stored: data type (numeric, currency, alphabetic, name, date, address) and any restrictions on the data input (number of characters; restricted to certain terms such as Mr., Ms. or Dr.; M or F).Structured data has the advantage of being easily entered, stored, queried and analyzed.[3]

B. Inverted Index

An inverted index(postings file or inverted file) is a data structure used for document retrieval. Inverted index have two variants :1)**record level inverted index**(inverted file index or inverted file) – tells you which document contains that particular word.2)**word level inverted index**(fully inverted index or inverted list)—tells you both which document contains that particular word and the place(position) of the word in that particular document. This can be represented as (document-id**;**no:of occurrences,[position]).For example

D_0– {this is to test a test}
Positions: 0 1 2 3 4 5
D_1—{this is testing to test application}
D_2—{this is a testing application}
The words would store in the inverted index like this:

Table 1:Inverted Index table

words	Inverted file ndex	inverted list
a	{0,2}	{(D_0;1,[4])(D_2;1,[2])}
application	{1,2}	{(D_1;1,[5])(D_2;1,[4])}
is	{0,1,2}	{(D_0;1,[1])(D_1;1,[1])(D_2;1,[1])}
test	{0,1}	{(D_0;2,[3,5])(D_1;1,[4])}

CONFERENCE PAPER
4th National Conference on Recent Trends in Information Technology 2015 on 25/03/2015
Organized by Dept. of IT, Prasad V. Potluri Siddhartha Institute of Technology, Kanuru, Vijayawada-7 (A.P.) India

testing	{1,2}	{(D_1;1,[2])(D_2;1,[3])}
this	{0,1,2}	{(D_0;1,[0])(D_1;1,[0])(D_2;1,[0])}
to	{0,1}	{(D_0;1,[2])(D_1;1,[3])}

C. Compression

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 form 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.

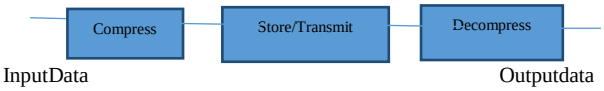

Figure1. Compression Procedure

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.
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.

II. BACKGROUND AND RELATEDWORK

The Jinlin Chen, Terry Cook[12] use a special data structure, Up Down Tree, to implement an efficient top-down approach for DSP (d-gap sequential patterns)mining. DSP information can be combined with Gamma codes for index compression The disadvantage of gamma code with d-gap is Encoding and decoding method is sequential, it seems hard to implement .It requires a lot of bit-shifting and masking, and they can be slow . the sequence of *d*-gaps (1,*l*, 1,*l*, 1,*l)* where l is a large number, has a very high degree of structure for a more principled compression mechanism to exploit, but is expensive and are better only for small numbers.Basic d-gaps coding methods are Variable Byte, which does not give the best compression performance

An entropy encoding is a lossless data compression scheme that is independent of the specific characteristics of the medium.One of the main types of entropy coding creates and assigns a unique prefix-free code to each unique symbol that occurs in the input. These entropy encoders then compress data by replacing each fixed-length input symbol with the corresponding variable-length prefix-free output codeword.

Two of the most common entropy encoding techniques are Huffman coding and arithmetic coding.[10]
Implementing arithmetic coding is more complex than Huffman coding. Huffman coding is faster than arithmetic coding. Arithmetic encoding is avoided because of patent issues. For Huffman encoding the redundancy is almost zero When more than two "symbols" in a Huffman tree have the same probability, different merge orders produce different Huffman codes so, we prefer code with a smaller length variance.
Different entropy coding techniques

A. Arithmetic Coding

1. It is a paid algorithm(protected by patent).
2. Statistical technique

B. Run length Coding

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.

C. LZW(Lempel–Ziv–Welch)

OutputData

1. Amount of storage needed is indeterminate.
2. It is a paid algorithm
3. Management of string table is difficult.

D. Huffman Coding

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 are 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]

E. Huffman Coding Advantages

1. Easy to implement
2. Simple technique

CONFERENCE PAPER
4th National Conference on Recent Trends in Information Technology 2015 on 25/03/2015
Organized by Dept. of IT, Prasad V. Potluri Siddhartha Institute of Technology, Kanuru, Vijayawada-7 (A.P.) India

3. Produces optimal and compact code
4. High speed
5. It is not a paid algorithm
6. Compression ratio for Huffman coding algorithm falls in the range of 0.57 to 0.81.
7. Given the Huffman tree, very easy (and fast) to encode and decode
8. When all the data is known in advance its better to use Huffman encoding.

When all the data is not known in advance its better to use LZW.Huffman coding and arithmetic encoding (when they can be used) give at least as good, and often better compression than any universal code.

However, universal codes are useful when Huffman coding cannot be used — for example, when one does not know the exact probability of each message, but only knows the rankings of their probabilities.[11]

These are some universal codes for integers

- Elias gamma coding
- Elias delta coding
- Elias omega coding
- Exp-Golomb coding
- Fibonacci coding
- Levenstein coding
- Byte coding, also known as comma coding

These are non-universal ones:

- unary coding, (which is used in Elias codes).
- Rice coding,
- Golomb coding, (which has Rice coding and unary coding as special cases).

Universal codes are generally not used for precisely known probability distributions, and no universal code is known to be optimal for any distribution used in practice.

Table II: Comparisons of different Encoding techniques

	Huffman	**Arithmetic**	**Lempel-Ziv-welch**
Probabilities	Known in advance	Known in advance	Not Known in advance
Alphabet	Known in advance	Known in advance	Not Known in advance
Data Loss	None	None	None
Symbols dependency	Not used	Not used	Not used
Code Words	One codeword for each symbol	One codeword for all data	Code words for set of data

III. EXPERIMENT EVALUATION

The algorithm for the Huffman coding is

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:
    1. Remove the two nodes of highest priority (lowest probability) from the queue
    2. Create a new internal node with these two nodes as children and with probability equal to the sum of the two nodes' probabilities.
    3. 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].A sample tree with two nodes will look like as:

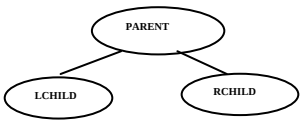

Fig 2: Sample Huffman tree

The flow chart for the Huffman coding algorithm is :

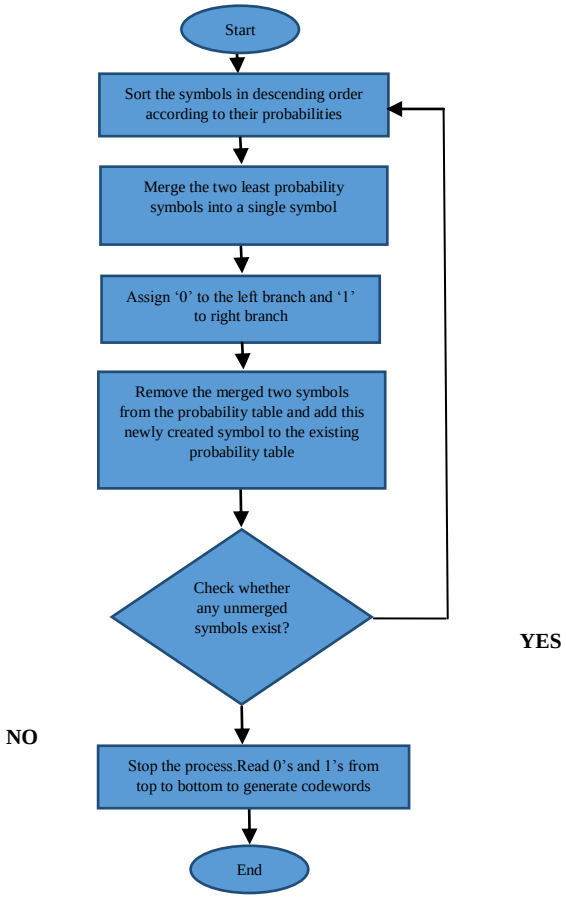

Figure 3 Flowchart for Huffman Encoding

IV. IMPLEMENTATION

The Huffman algorithm is implemented by taking the example as

CONFERENCE PAPER
4th National Conference on Recent Trends in Information Technology 2015 on 25/03/2015
Organized by Dept. of IT, Prasad V. Potluri Siddhartha Institute of Technology, Kanuru, Vijayawada-7 (A.P.) India

Eg: this is to test

Characters

Frequency	**4**	**3**	**3**	**2**	**1**	**1**	**1**

Figure 4.Characters and corresponding frequencies(- is used for space)

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 '**h**' 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.

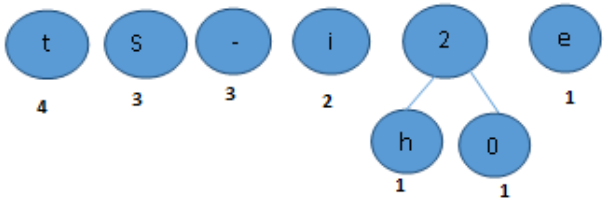

Figure 5. Combining the least two frequencies(h and o) as a single node

Repeat that step,choose the next two minimal nodes,it must be e and the combined tree of h and o.The collection of nodes shrinks by one each iteration.we remove two nodes and add a new one back in.

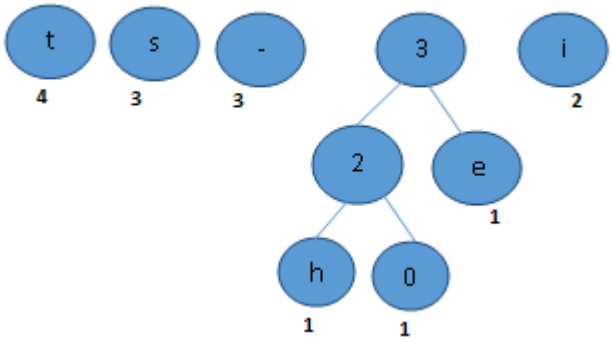

Figure 6: Combining the least two frequencies(e and combined tree of h and o) as a single node

Again, we pull out the two smallest nodes and build a tree of weight 5:

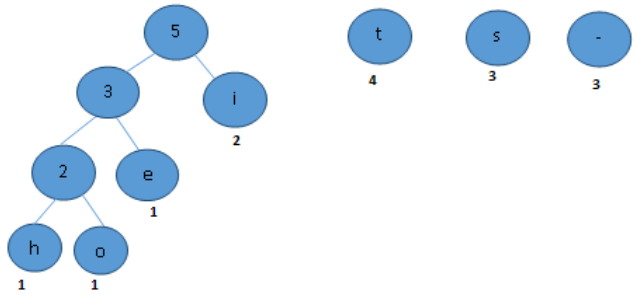

Figure 7: Combining the least two frequencies(i and combined tree of h ,o and e) as a single node

Build a combined node from s and –(space).The interior nodes are used along the paths that eventually lead to valid encodings, but the prefix itself does not encode a character.

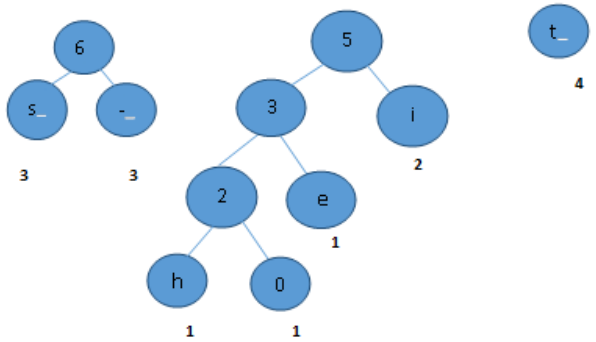

Figure 8: Combining the least two frequencies(s and –(space)) as a single node

Again, we pull out the two smallest nodes(5and t) and build a tree of weight 9:

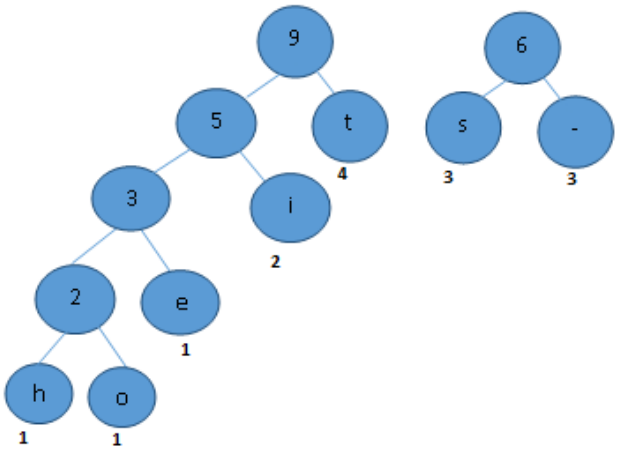

Figure 9: Combining the least two frequencies(t and combined tree of h,o,e and i) as a single node

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

CONFERENCE PAPER
4th National Conference on Recent Trends in Information Technology 2015 on 25/03/2015
Organized by Dept. of IT, Prasad V. Potluri Siddhartha Institute of Technology, Kanuru, Vijayawada-7 (A.P.) India

the number of characters in the input file.

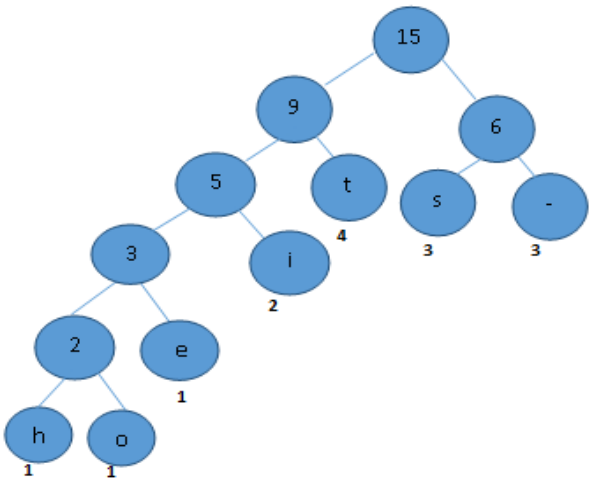

Figure 10: Final Huffman tree

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

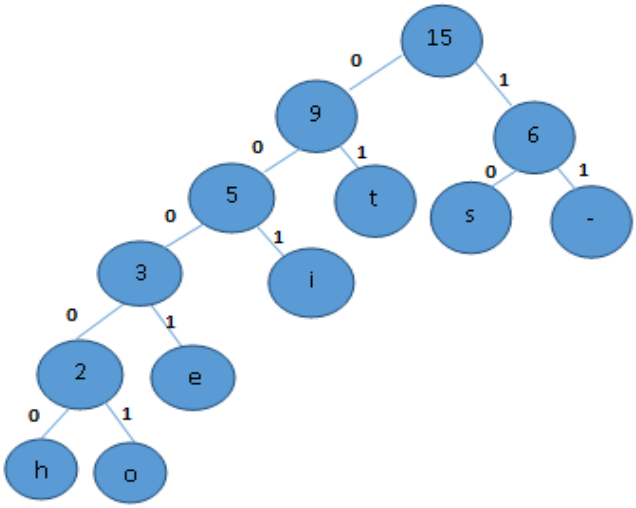

Figure 11: Huffman Tree with 0's and 1's

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

characters	ASCII	ASCII Encoding (fixed length)	Huffman Encoding (variable length code)
t	116	01110100	01
s	115	01110011	10
-(space)	32	00100000	11
i	105	01101001	001
e	101	01100101	0001
h	104	01101000	00000
o	111	01101111	00001

Using the standard ASCII encoding, this 15 character string requires 15*8 = 120 bits total. But with Huffman coding technique this 15 characters require only 40 bits .The string **"this is to test"** encoded using the above variable-length code table is:

t	h	i	s		i	s		t	o		t	e	s	t
01	00000	001	10	11	001	10	11	01	00001	11	01	0001	10	01

V. CONCLUSION

This paper describes the inverted index compression over structured data. Compression can greatly reduce the number of disk accesses, with the help of Huffman coding, overhead costs are much reduced and greatly increases the query evaluation speed, When an index fits in main memory, the transfer of compressed data from memory to the cache is less than that of transferring uncompressed data. Experiments results shown that inverted index compression saves storage space and also increases the keyword search speed. We implemented Huffman coding technique on characters ,for the extension of this paper will be implemented by using the Huffman coding to the sequence of characters i.e words and then perform the index compression.

VI. ACKNOWLEDGEMENTS

We like to express our gratitude to all those who gave us the possibility to carry out the paper.We would like to thank to **REV.Fr.J.Thainese S.J ,Director , ALIET** for stimulating suggestions and encouragement.

VII. REFERENCES

[1] http://en.wikipedia.org/wiki/Search_engine_indexing#The_forward_index
[2] http://googleblog.blogspot.in/2008/07/we-knew-web-was-big.html
[3] http://www.webopedia.com/TERM/S/structured_data.html
[4] http://en.wikipedia.org/wiki/Inverted_index
[5] http://en.wikipedia.org/wiki/Data_compression
[6] David Solomon ",Data compression, The complete Reference", Fourth edition, Springer
[7] Khalidsayood," Introduction to data compression", Third edition
[8] M.Nelson,J.L.Gailly," The Data Compression Book", second edition
[9] http://en.wikipedia.org/wiki/Huffman_coding
[10] http://en.wikipedia.org/wiki/Entropy_encoding
[11] http://en.wikipedia.org/wiki/Universal_code_(data_compression)
[12] J. Chen,T. Cook" Using d-gap Patterns for Index Compression",2007, ACM.

CONFERENCE PAPER
4th National Conference on Recent Trends in Information Technology 2015 on 25/03/2015
Organized by Dept. of IT, Prasad V. Potluri Siddhartha Institute of Technology, Kanuru, Vijayawada-7 (A.P.) India