A Novel Approach of Data Compression by Modified Huffman Tree

Satyendra Nath Mandal¹, Subhadip Ghosh², Goutam Gorai³, Mita Layek⁴ and Debjani Saha⁵

¹Asst. Professor, Kalyani Govt. Engg. College, Kalyani, Nadia(W.B).
¹satyen_kgec@rediffmail.com and ⁵debjani89@gmail.com

ABSTRACT
Nowadays Huffman data compressing algorithm is used in many data compression applications. In this paper, we analyze and compare the compression ratio between our implemented code and the basic tree representation of Huffman algorithm. Here we have tried to reduce the space while sending the compressed data as a single stream of character.

KEYWORDS
Huffman Data Compression Algorithm, Lossless Data Compression, Compression Ratio.

1. INTRODUCTION
In the last decade we have experienced marvelous growth in the Information technology innovations and applications. Information Technology has become a vital component for the success of business because most of the organizations require fast and less occupied space information dissemination, information processing, storage and retrieval of data. The growth in this area occurred at such a fast rate due to the fact that Information Technology [1][2] opened new vistas in almost day –to-day problems related with common man. Information Technology has revolutionized our life and has made a significant impact in all dimensions of our day-to-day life. In banking sector, use of credit , debit card , ATM, Tele-banking, Net banking[6]; in transportation ,reservation of air tickets[5], buying and selling items on internet, electronic market, inquiry of department, bank transaction on net, entertainment, education, communication, hotel reservation[3], tourism have become reality. Internet is one of the mediums, which being used to access the pool of information.
Proper transformation of data is main theme in this era. Sometimes information becomes so large that it becomes problematic to transmit or storing information in their proper format. So the concepts of data compression arise. That means, transformation of information in certain format which will take much small space comparing to original data. Shannon-Fanon algorithm [7], Huffman algorithm [4] & Arithmetic Coding [8] are some process of data compression. The works include “Optimal Huffman Tree –Height Reduction for Instruction Level Parallelism”, Dept of Computer Science, the University of Texas at Austin reports on a work of exploiting instruction level parallelism (ILP) is a key component of high performance for modern processor. For this purpose, Huffman Algorithm was taken to (1) tree height reduction rewriting expression trees of the commutative and associative operations to make the height of the tree reduced (2) software fan-out generating software to fan out tree even when expression store intermediate of the instruction. [8]

Another work for the concurrent update and generation of the dynamic Huffman code on is made for the dynamic Huffman Encoding. The concurrent procedure performs the tree update and code generation process in parallel and therefore reduced over 45% number of steps required by the Knuth’s work [9]. Work on the fast Adaptive Huffman Encoding Algorithms state that Huffman code suffers from two problems: the prior knowledge of the probability distribution of the data store to be encoded is necessary, and the encoded data propagates errors. The first problem can be solved by an adaptive coding, when the second problem can be solved partly by segmenting data into segments. But the adaptive Huffman code performs badly when segmenting data into segments. But the adaptive Huffman code performs badly when segmenting data into relatively small segments because of its relatively slow adaptability [7]

2. ALGORITHM
We are representing the TREE in a string of ‘0’ and ‘1’.
The algorithm is very simple:
1.0 Start with the root node.
2.0 Check if it is a leaf node or not.
2.1 If it is a leaf node then add ‘1’ at the end of the string and then the ASCII code of the corresponding letter written in the node.
2.2 Else the current node is an intermediate node then add ‘0’ at the end of the string and push first the left child and then the right child in the queue.
2.3 Pop a node from the queue and repeat step 2.0 to 2.3 until the queue is empty.

Now we have the string just having a sequence of ‘0’ and ‘1’ representing the Huffman Tree.

Regenerating the tree from the string:
1.0 Make a node (it will be the root node of the TREE).
2.0 Take the first element of the string
2.1 Check whether it is ’0’ or ’1’.
2.2 If it is ’1’ (that means the present node is a leaf node) take the next 8 elements of the string and convert it in the character form and put it in the node.
2.3 If it is ’0’ (that means the current node is an intermediate node) make two child nodes of the current node.
and push first the left child and then the right child into the queue.

2.4 Pop a node from the queue. Take the next element of the string. Repeat step 2.1 to 2.4 until the queue is empty.

3. Experimental Result

Compression ratio between earlier method and our method:

Space required for representing the TREE (i.e. the frequency table) - space required by the string representing the TREE)*100/Space required for representing the TREE (i.e. the frequency table)

The data we got from taking text files of different size:

In the earlier method the frequency table of the text was being sent along with the text for reconstructing the TREE. The Frequency table required at least two fields, one containing the character and another containing the corresponding frequency.

A char takes say, 1 byte = 8 bits
An int takes say, 2 bytes = 16 bits

If there is ‘n’ no of characters present in the text total

(8+16)*n=24 n no of bits.

In our algorithm we need (2*n-1 ) no of bits to represent the tree and (8*n) bit for the characters.(assuming that a character is being represented with 8 bits).

Therefore the total no of bits required is ( 2*n-1+8*n)=(10*n)-1

Compression ratio between earlier method and our method:

((24*n)-(10*n-1)))*100/24*n = 58.333

CONCLUSION AND FUTURE WORK

So by going through our implemented algorithm you will see that we do not need to send two files at all. So we don’t have any headache for linking two files, subsequently no header and separator is used. Whereas previously header was used for linking, supplying extra bit if necessary and keeping required information regarding this. Our next target will be to look into reducing the time complexity more so we need to concentrate on further improvement on our methods/techniques.

REFERENCES


