Huffman Tree and Lossless Data Compression Using Huffman Method and Comparison with Other Technique

Wasim FirujAhmed[1], Sanjukta[2]
[1,2]Department of computer science, VIT University, Vellore, India

Abstract— This paper investigates and shows us how to create a tree from leaf node. Huffman tree is built from bottom up rather than top down that is creation of tree starting from leaf node and proceed upwards. Also this paper shows us the application of Huffman tree that is in lossless data compression and it is compared with other compression techniques. I have analyzed Huffman and compared it with arithmetic coding. Data compression is a process that reduces size of the data which is useful as cost will be reduced and redundancy in data is also reduced. Data compression is used in distributed system. So for speed and performance efficiency data compression is used there.

Key words: Huffman, lossless, lossy, run length encoding, arithmetic coding

I. INTRODUCTION

In Huffman tree the creation of tree starts from leaf node and proceeds upwards to get the root node .It is an extended binary tree with minimum weighted external path length for the external nodes.it finds its main application in data compression. Data compression refers to reducing size of the given data used to represent image or video or a file. Now question comes as why we should reduce the data. The answer is quite simple: to reduce the cost, to make optimal use of limited storage space and to save the time. There are different techniques to reduce data size and everyone have their own advantage and disadvantage. Some of them are Huffman coding, Run Length encoding, Arithmetic encoding. But the best one can be understood only through comparison. Run length encoding method is frequently applied to images or pixels in a scan line. It is a small compression component used in JPEG compression. Arithmetic coding is a technique for lossless data compression. It is a form of variable-length entropy encoding. Arithmetic coding encodes the entire message into a single number, a fraction n where ($0.0 \leq n < 1.0$) but where other entropy encoding techniques separate the input message into its component symbols and replace each symbol with a code word.

II. WORKING OF HUFFMAN TREE

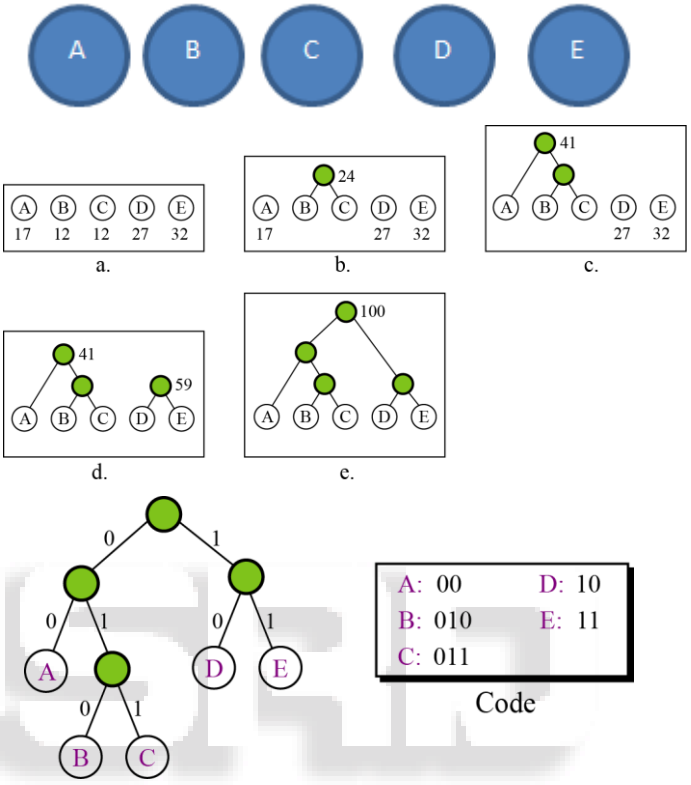
Huffman tree is built from bottom up rather than top down i,e creation of tree starting from leaf node and proceed upwards.

A. *Steps to build Huffman Tree*

1) Create a leaf node for each unique character
2) Extract two nodes with the minimum frequency
3) Create a new internal node with frequency equal to the sum of the two nodes frequencies.
4) Repeat steps#2 and #3 until the heap contains only one node. The remaining node is the root node and the tree is complete

B. *Example:*

Character	A	B	C	D	E
Frequency	17	12	12	27	32



III. DATA COMPRESSION

A. *Types of compression*

- Lossy compression: here some data is lost after decompression.
- Lossless data compression: The name lossless means "no data is lost". When the data is decompressed, the result perfectly matches with the original one. Here data is saved efficiently in compressed state and no data is lost after decompression.

B. *Compression methods*

1) Run length encoding: is a very simple form of data compression in which runs of data (that is, sequences in which the same data value occurs in many consecutive data elements) are stored as a single data value and count, rather than as the original run. This is most useful on data that contains many such runs: for example, relatively simple graphic images such as icons, line drawings, and animations. It is not useful with files that don't have many runs as it could potentially double the file size.
For example, if the input string is "wwwwaaadexxxxxx" , then the function should return "w4a3d1e1x6″.

Algorithm:

(1) Pick the first character from source string.
(2) Append the picked character to the destination string.
(3) Count the number of subsequent occurrences of the picked character and append the count to destination string.
(4) Pick the next character and repeat steps b) c) and d) if end of string is NOT reached

This algorithm is very easy to implement and does not require much CPU horsepower. RLE compression is only efficient with files that contain lots of repetitive data.

Time complexity: O(n)

2) Arithmetic encoding

Arithmetic coding (AC) is a special kind of entropy coding. Unlike Huffman coding, arithmetic coding doesn´t use a discrete number of bits for each symbol to compress .The biggest drawback of the arithmetic coding is it´s low speed since of several needed multiplications and divisions for each symbol. The main idea behind arithmetic coding is to assign to each symbol an interval. Starting with the interval [0..1], each interval is divided in several subintervals, which sizes are proportional to the current probability of the corresponding symbols of the alphabet. The subinterval from the coded symbol is then taken as the interval for the next symbol. The output is the interval of the last symbol. For Implementations write bits of this interval sequence as soon as they are certain.

Let us take an example for this:

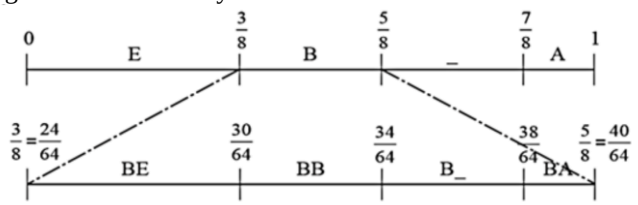
BE_A_BEE

And we now compress it using arithmetic coding.

Step 1: in the first step we do is look at the frequency count for the different letters:

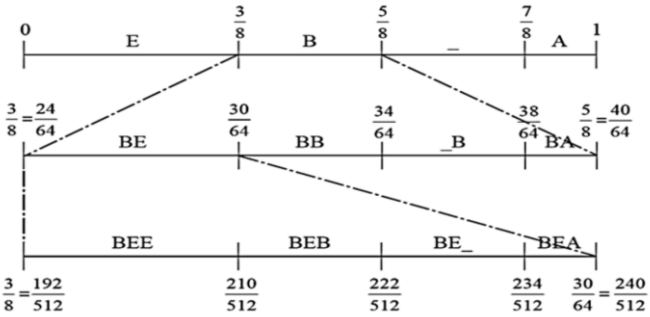
E-3; B-2; _-2; A-1

Step 2: In second step we encode the string by dividing up the interval [0, 1] and allocate each letter an interval whose size depends on how often it count in the string. Our string start with a "B", so we take the „B" interval and divide it up again in the same way:

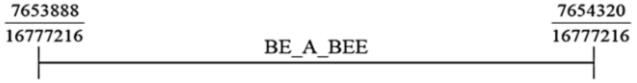


The boundary between „BE" and „BB" is 3/8 of the way along the interval, which is itself 2/3 long and starts at 3/8. So boundary is 3/8 + (2/8) * (3/8) = 30/64. Similarly the boundary between „BB" and „B_" is 3/8+ (2/8) * (5/8) = 34/64, and so on.

Step 3: In third step we see next letter is now „E", so now we subdivide the „E" interval in the same way. We carry on through the message….And, continuing in this way, we eventually obtain:



And if we continue like this we will get;



So we represent the message as any number in the interval [7653888/16777216, 7654320/16777216].

But, we cannot send numbers like 7654320/16777216 easily using computer. In decimal notation, the rightmost digit to the left of the decimal point indicates the number of units; the one to its left gives the number of tens: the next one along gives the number of hundred, and so on. So 7653888 = (7*106) + (6*105) + (5*104) + (3*103) + (8*102) + (8*10) + 8 Binary numbers are almost exactly the same, only we deal with powers of 2 instead of power of 10. The rightmost digit of binary number is unit (as before) the one to its left gives the number of 2s, the next one the number of 4's, and soon. So 110100111 = (1*28) + (1*27) + (0*26) + (1*25) + (0*24) + (0*23) + (1*22) + (1*21) + 1 = 256 + 128 + 32 + 4 + 2 + 1 = 423 in denary (i.e. base 10).

3) Huffman coding

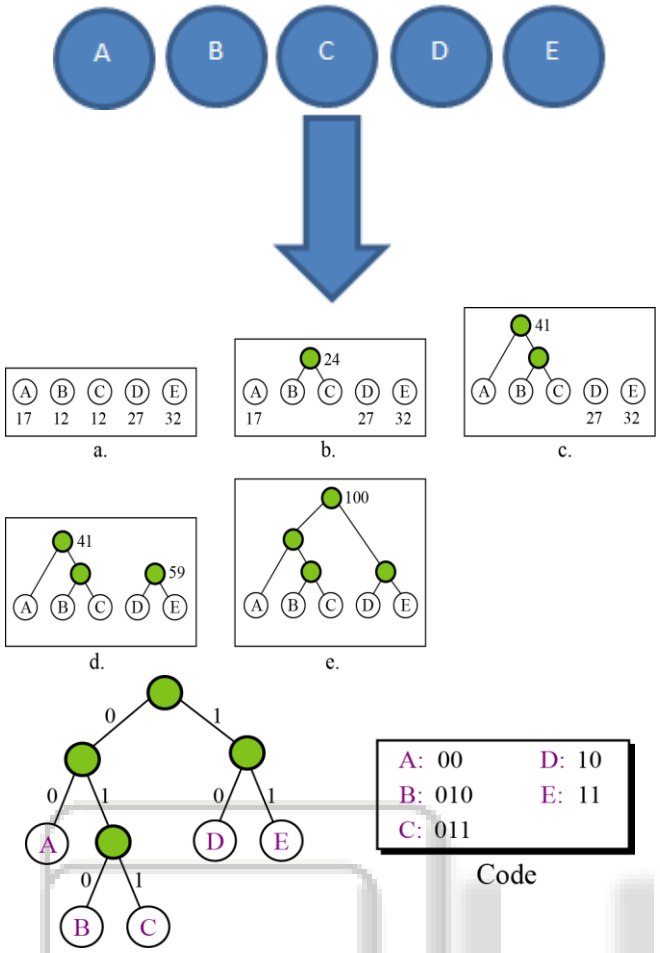
Huffman coding is an efficient compression technique for lossless data compression in computer science and information theory. The term refers to the use of a variable-length code table for encoding a source symbol (such as a character in a file) where the variable-length code table has been derived in a particular way based on the estimated probability of occurrence for each possible value of the source symbol.in this technique we assign fewer bits to symbol that occur more frequently and lesser to symbol that occur less frequently. There is no unique Huffman code and every Huffman code has same average code length.

- Algorithm:

a) Make a leaf node for each code symbol
   Add the generation probability of each symbol to the leaf node
b) Take the two leaf nodes with the smallest probability and connect them into a new node
   Add 1 or 0 to each of the two branches
   The probability of the new node is the sum of the probabilities of the two connecting nodes
c) If there is only one node left, the code construction is completed. If not, go back to step (2).

Example:

character	A	B	C	D	E
frequency	17	12	12	27	32

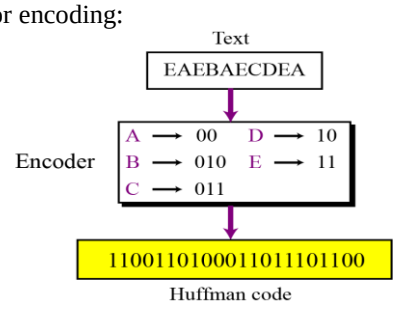


Above tree is the Huffman tree.

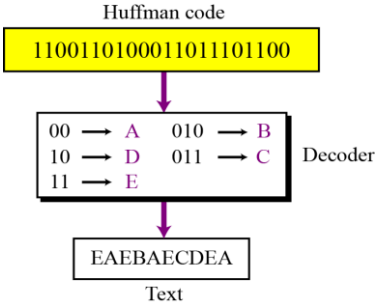
The same Huffman tree is used for decoding data. The encoded data is read bit by bit from left side. We will start from root, if there is a 0 in the coded message we go to left child and if there is 1 we go to right child and this process continues till we reach the leaf node. On reaching the leaf node we get the symbol and again we start from root node.

Let us take a coded message and decode it.

- For encoding:



- For decoding:



4) Compression performance

Performance measure is use to find which technique is good according to some criteria. Depending on the nature of application there are various criteria to measure the performance of compression algorithm. When measuring the performance the main thing to be considered is space efficiency [5]. And the time efficiency is another factor. Since the compression behavior depends on the redundancy of symbols in the source file, it is difficult to measure performance of compression algorithm in general. The performance of data compression depends on the type of data and structure of input source.

- **Compression ratio:** compression ratio is the ratio between size of compressed file and the size of source file.

  Compression ratio = (size after compression/size before compression)

Compression factor: compression factor is the inverse of compression ratio. That is the ratio between the size of source file and the size of the compressed file.

Compression factor = (1/compression ratio)

Saving percentage calculates the shrinkage of the source file as a percentage.

Saving percentage = ((size before compression - size after compression)/size before Compression)) %

	arithmetic	**Huffman**
Compression ratio	Very good	poor
Compression speed	slow	fast
Decompression speed	slow	fast
Memory space	Very low	low
Compressed pattern matching	no	yes
Advantage	1. Efficiently represents more frequently occurring sequences of pixels values with fewer bits. 2. Reduce file size dramatically. 3. Lossless compression.	1.Easy to implement 2. Lossless technique 3. Produces optimal and compact code
Disadvantage	1. It is a paid algorithm (protected by patent). 2. Statistical technique.	1. Relatively slow. 2. Depends upon statistical model of data. 3. Decoding is difficult due to different code lengths.

Table 1: comparison between compression techniques

IV. CONCLUSIONS

I have studied various techniques for compression and compare them on the basis of their use in different applications and their advantages and disadvantages. I have concluded that arithmetic coding is very efficient for more frequently occurring sequences of pixels with fewer bits and reduces the file size dramatically. RLE is simple to implement and fast to execute. Huffman algorithm is used in JPEG compression. It produces optimal and compact code but relatively slow. Huffman algorithm is based on statistical model.

REFERENCES

[1] http://en.wikipedia.org/wiki/Lossless_JPEG
[2] http://en.wikipedia.org/wiki/Arithmetic_coding
[3] http://en.wikipedia.org/wiki/David_A._Huffman
[4] The data compression book by Mark nelson and jean-loup Gailly
[5] www.slideshare.net

