Huffman coding in data compression

Md. Ismail Nabonitta Biwsas
Roll No:1503094 Roll No:1503095
Department of Computer Science and Engineering
Rajshahi University of Engineering and Technology

***Abstract*—Huffman coding is a greedy lossless data compression algorithm. This technique use variable length encoding. Encoding also satisfies prefix rule of encoding. In this paper we will show mathematically how Huffman coding compress data such as text file. We also compare Huffman data compression technique with ASCII system.**

I. Introduction

A. *Data compression:*

Generally data compression is modifying the data in such a way that resulting modified data take less space than input data.It includes a stream of symbols and transforming them into codes[**nelson1995data**]. Data-compression techniques can be divided into two major families.[**nelson1995data**]

1) Lossy data compression
2) Lossless data compression

B. *Lossy and lossless data compression:*

Difference between lossy data compression and lossless data compression is given bellow:

TABLE I
Difference between lossy data compression and losless data compression

Lossy data compression	Lossless data compression
Additional and unnecessary data is discards from input data[**nelson1995data**]	No information is discarded[**nelson1995data**]
Used where where perfect reproduction is not required[**ng1997lossless**]	Used where perfect reproduction is required[**ng1997lossless**]
Applied in 3D data transferring, multimedia file compression such as JPEG file,MP3 file,MP4 file, digitized voice recognition[**nelson1995data, ng1997lossless**]	Data storing in database ,spreadsheets,word processing files. ,text file compression, compress medical images[**nelson1995data, ng1997lossless**]

In ASCII (American Standard Code for Information Interchange) each character is represented by eight bits of 0s and 1s binary string.But in Huffman encoding technique each character represent by different length of binary string in such a way that more common symbols are represented with fewer bits.[**han2015deep**] Generally there are two types of encoding system.

1) Fixed length encoding
2) Variable length encoding

Fixed length means that length of the code for any character is fixed where variable length encoding means that length of the code is variable.Variable length encoding is mere better than fixed length encoding.[**cormen2009introduction**]

C. *Prefix rule:*

In Huffman data compression technique bit string that represent a symbol or character is not prefix of the bit string which represent others symbol.[**cormen2009introduction**]Which is known as prefix rule.As a result ambiguity is removed during decoding time.
Example: If we find the set of bit string after decoding

$$\{t = 0, u = 11, e = 100, r = 101\}$$

is a prefix code.Now by encoding the bit string ”0100101101” we find only ”terr”.This can be illustrated in the figure.

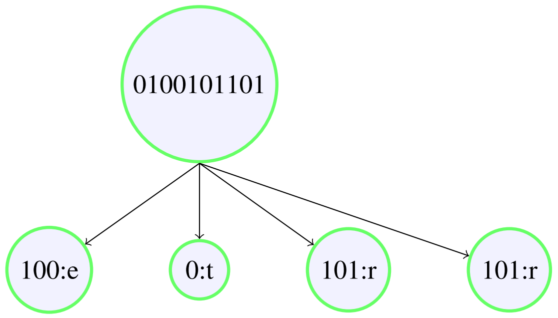

Fig. 1. Tree representation of prefix rule

But the bit string set

$$\{x = 0, y = 1, z = 11\}$$

is not a prefix code because the string ”111” can be represented as ”yyy” or ”yz” or ”zy”.This can be illustrated in the figure.

We organized our paper as follow. In sectionIII Huffman algorithm will be discussed.Experimental result with some mathematical calculation will be demonstrated in section IV. In this section we also compare Huffman coding with ASCII coding ,Arithmetic coding.Finally in section V advantages and disadvantages of Huffman coding will be mentioned.

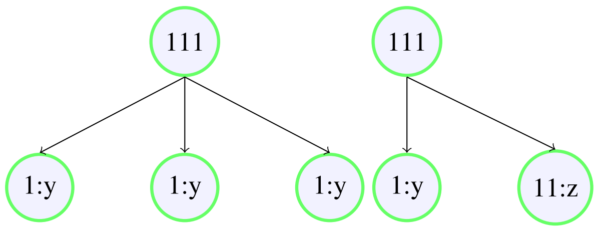

Fig. 2. Tree representation not a prefix rule

II. Background

When transferring a message of data or text ,if transmission time for each symbol is same then transmission time of message is directly proportional to length of message code.[**huffman1952method**].For this reason transferring message use ASCII and EBCDIC code take a long time also a huge amount of empty memory whereas variable length code need $\lceil log_2 n \rceil$ bit per letter[**vitter1987design**].David A. Huffman publish a paper where he propose a coding system based on the probability of character in message.

III. Methodology

Generally there are two steps of Huffman coding.They are:

- Constructing a Huffman tree from input data of character
- Traversing the Huffman tree and determining the code of the character

A. *Constructing a Huffman tree:*

Huffman tree can be built from the frequency of the character.Which is known as statistical modeling.[**nelson1995data**]Following are the process[**sharma2010compression**] of constructing a Huffman tree by using min heap:

i Make all the character leaf node of tree.
ii Push all the character into min priority queue i.e., characters are in ascending order of their frequency.
iii Pop first two element of priority queue and push their (first two element) sum as the top of priority queue.
iv Repeat step 3 till priority queue contain more than one node.
v Now tree is generated and remaining node is the root node of the Huffman tree.

Huffman tree are built from the bottom up[**nelson1995data**]. Algorithm 1 is the Huffman tree construction algorithm:[**cormen2009introduction**]

Algorithm 1 Huffman(no of character)

```
$n \leftarrow$ no of character
for $i = 0$ to $n - 1$ do
    $minheap\ .\ push$ (character,frequency)
    $i \leftarrow i - 1$
end for
while $minheap\ .size() > 1$ do
    create a new node N
    $N.left \leftarrow$ top of min heap
    pop min heap
    $N.right \leftarrow$ top of min heap
    pop min heap
    $N.top \leftarrow N.left + N.right$
    $minheap\ .push(top)$
end while
return  min heap
```

To determine the code of character we traverse the Huffman tree by using a recursive function. Algorithm 2 is the algorithm of traversing Huffman tree.

Algorithm 2 Traverse(root node,code)

```
if root is null then
    return
end if
if symbol of root is not null then
    print code
end if
Traverse(left child of root,$code + 0$)
Traverse(right child of root,$code + 1$)
```

Decoding code is very easy.We use STL to decode the text file. Following algorithm 3 is the pseudo code for decoding.

Algorithm 3 Pseudo code of decoding

```
$s \leftarrow$ input encoding string
$n \leftarrow$ no of individual character
declare map$< char, string >$code
for $i = 0$ to $n - 1$ do
    code.insert($make\_pair(symbol, code)$)
end for
declare string $f$ as NULL to store decode string
declare temporary string $x$ as NULL
for $i = 0$ to length of $s$ do
    $x \leftarrow x + s[i]$
    declare an iterator $it = code.begin()$
    while $it! = code.end()$ do
        if $it- > second == x$ then
            $f = f + it- > first$
            break
        end if
        $it \leftarrow it + 1$
    end while
end for
print $f$
```

Complexity analysis: For n nodes pop operation into while loop of 1 will execute (n-1) times.Each of loop take complexity of O(logn). So complexity of Huffman tree constructing algorithm is O(nlogn).

IV. Experiment

Example: A file containing frequencies shown at table II.Determine the Huffman code of each symbol.

Solution: Now construct Huffman tree step by step.Figure 3 to figure 7 illustrate different steps of above example. step1:

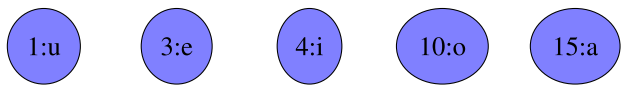

Fig. 3. Huffman tree after first pass

TABLE II
SYMBOL WITH FREQUENCIES

Symbol	Frequencies
a	15
e	3
i	4
o	10
u	1

Step2:

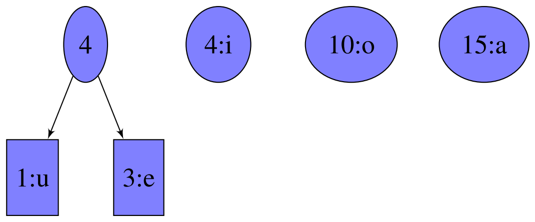

Fig. 4. Huffman tree after second pass

step3:

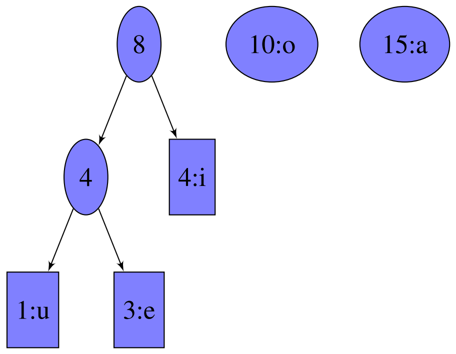

Fig. 5. Huffman tree after third pass

Step4:

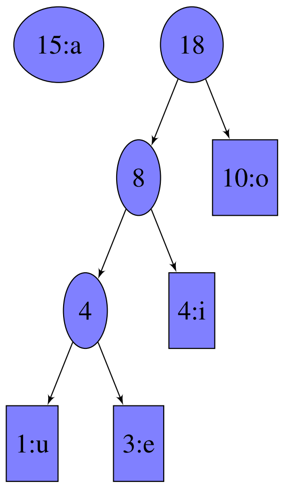

Fig. 6. Huffman tree after forth pass

Step5: In this step Huffman tree is created.Bit ’0’ represent the left child and bit ’1’ represent right child.[**sharma2010compression**]

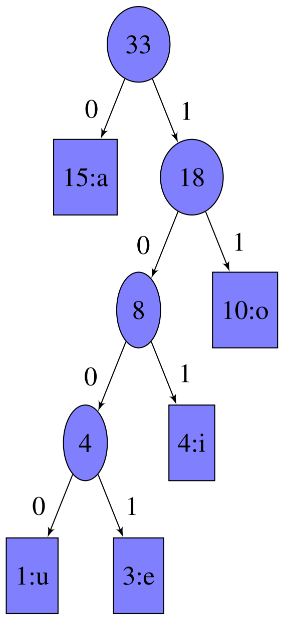

Fig. 7. Final Huffman tree

To determine the Huffman code of each symbol traverse Huffman tree starting from the root node to the every leaf node.Code of each symbol is tabulated at table III.

From table III it is proved that the length of code is variable whereas length of ASCII code is fixed.Average length of ASCII code is 8.Average length of Huffman code is determined by using formula 1[**cormen2009introduction**] and the number of bits in encoded message is determined by formula 2[**cormen2009introduction**].Entropy of Huffman can be determined by using formula 3.[**howard1994arithmetic**]

TABLE III
SYMBOL WITH THEIR CODE

Symbol	Huffman Code	Length of Huffman code	ASCII code	Length of ASCII code
a	0	1	01100001	8
o	11	2	01101111	8
i	101	3	01101001	8
e	1001	4	01100101	8
u	1000	4	01110101	8

$$average\ length\ of\ code = \frac{\sum(frequency_i \ \times \ code\ length_i)}{\sum frequency_i} \tag{1}$$

number of bits in encoded message,

$$= (\sum number\ of\ character\ \times\ average\ length\ of\ bit) \tag{2}$$

$$H(x) = \sum_{k=1}^{k=n} -p_k log_2 p_k \tag{3}$$

For Huffman code of table III average length of code ,

$$L = \frac{(15 * 1) + (10 * 2) + (4 * 3) + (3 * 4) + (1 * 4)}{(15 + 10 + 4 + 3 + 1)}$$

$$= 1.909$$

As Huffman coding use integral number of bit of code word it is 2.[**nelson1995data**] Hence percentage of compression of data by using Huffman coding,

$$= \frac{8-2}{8} \times 100$$

$$= 75$$

TABLE IV
PROBABILITY TABLE OF TABLEIII

Symbol	Length of Huffman code	p_k	$log_2 p_k$	$p_k log_2 p_k$
a	1	0.4545	-1.1376	0.51703
o	2	0.0909	-1.0414	0.0946
i	3	0.1212	-0.9164	0.11106
e	4	0.3030	-0.51855	0.1572
u	4	0.030030	-1.5122	0.04541
				$\sum = .9253$

Now by using data from table IV and formula 3 find the entropy ,

$$H(x) = 0.9253$$

Hence the percentage of efficiency of the Huffman coding data compression,

$$\eta = \frac{H(x)}{L}$$

$$= \frac{0.9253}{1.909} \times 100 = 48.45$$

A. *Result analysis:*

In our calculation we show that efficiency is almost 50 percent and it can compress 75 percent than normal ASCII coding.In generally it varies from 20 percent to 90 percent which is depend on the property of data being compressed.[**cormen2009introduction**]

V. CONCLUSION

Main advantages of Huffman coding is it is easy to implement.But it depends on statistical model of data.If there is a wrong in the statistical data it does not give correct compression.Although Huffman coding is better than ASCII coding , it has some demerits. It uses integer number of code length.So it is does not give good data compression.Arithmetic coding data compression technique can remove this.