DAIICT

Gandhinagar

Batch '24

Academic Year 2020-21

SC-205 PROJECT

END-SEM Project

-Made by combined efforts of:

Suyash Vyas - 202001002

Praneet Shroff - 202001004

Vihar Shah - 202001110

Rushabh Patel - 202001419

July, 2021

HUFFMAN CODING

Assigned By:
Professor Manish Gupta
Professor Rahul Muthu

Abstract

In the corresponding report, we have tried to explore and implement one of the most efficient and trending worldwide method of source encoding named as Huffman coding and even focussed on its usability and application in real life.

1 Introduction

The source coding concept is very well discussed topic across the globe where researchers are constantly putting their efforts to make the process data efficient and cost-effective at the same time. Even in one of courses of Communication - Introduction to Communication System, we were taught regarding the data transfer, encoding and decoding at the transmitter and receiver end respectively. Thus, it was quite fascinating for us to deep dive into it and code a program which uses the concept of mathematics in order to solve the algorithm and thus execute one of the best known and most compressible form of the data encoding.

2 The Formal Statement of the Problem

The encoding process can be done for all characters (alphabets, numeric, Greek symbols) with the help of normal decoding, i.e., by assigning a codeword for every unique character and then transmit it, e.g., assuming that there are 6 unique characters. Thus, the codeword used for transmission of each alphabet would comprise a total of 3 binary digits. Let's write the codeword for few of them

$X1 = 000$
$X2 = 001$
$X3 = 011$
$X4 = 100$
$X5 = 101$
$X6 = 110$

Thus, during the transmission of a string of 100 characters, we would require a total of 3*100 = 300 bits to be transferred, with an average of 3 bits per character.

So, in order to reduce the number of bits transfer, we use the concept of Huffman Coding which is an algorithm that takes as input the frequencies (which are the probabilities of occurrences) of symbols in a string and produces as output a prefix code that encodes the string using the fewest possible bits, among all possible binary prefix codes for these symbols.

In simple words it makes codewords with respect to the frequency of occurrence, since more frequent characters have shorter codewords, the bits require to transmit the data will eventually decrease.

3 What is Huffman Coding?

A MIT graduated scientist David Huffman created an algorithm to compress the data by using the Greedy-choice property and Optimal substructure. The algorithm that takes the frequency of symbols (probability of occurrences) and produces as output a prefix code* that encodes the input string in lowest possible bits.

An example of Huffman coding is illustrated below:

Symbol	Probability / Frequency	Codeword	Length	Product
$X1$	0.45	0	1	0.45
$X2$	0.2	111	3	0.6
$X3$	0.13	110	3	0.39
$X4$	0.1	100	3	0.3
$X5$	0.07	1011	4	0.28
$X6$	0.05	1010	4	0.2

Here we can clearly observe that the sum of the product column is
$0.45 + 0.6 + 0.39 + 0.3 + 0.28 + 0.2 = 2.2$

This indicates the average number of bits required which is less than 3 bits which would be consumed as seen before method.

* Prefix code is a property where no bit string corresponds to more than one sequence of letters is to encode letters so that the bit string for a letter never occurs as the first part of the bit string for another letter.

e.g. - $x1 = 1, x2 = 00, x3 = 01, x4 = 10$

so here the prefix code property is violated and thus it will create errors while decoding, i.e., for an output of 001001 we can have multiple possibilities of the input code, it can be $x2x1x2x1$ or $x2x4x3$.

4 The Mathematics behind Huffman Coding:

This algorithm uses the concept of a data structure called "Binary Trees" to be more precise "min Heaps".

Rooted Binary Tree – It is a type of graph which have a rode node which acts as the source for a maximum of two branches, left subtree and right subtree. The subtrees can further be grown recursively to produce a huge binary tree / forest, comprising of multiple branches and multiple nodes.

Heap – It is a complete binary tree, which means that the tree is completely filled on all levels except possibly the lowest, which is filled from the left up to a point.

Thus, for the Huffman coding the following algorithm is implemented:

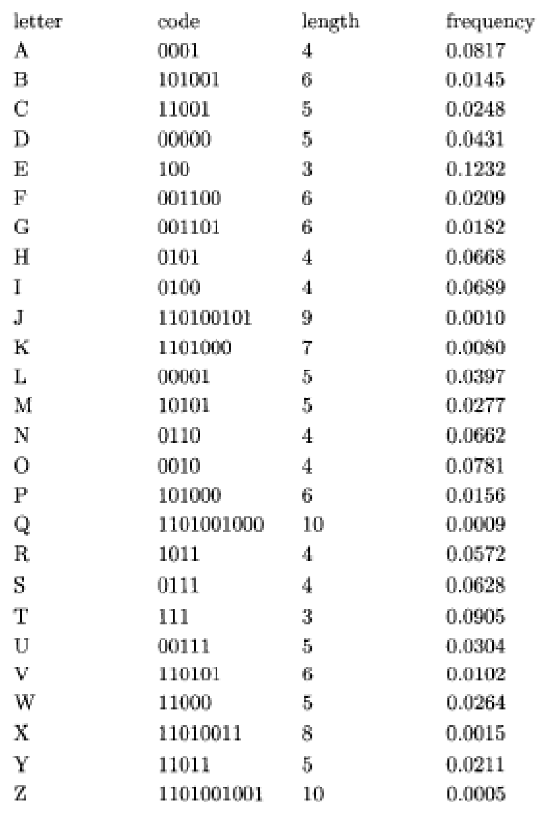
We assume that C is a set of n characters and that each character $c \in C$ is an object with an attribute c.freq giving its frequency. The algorithm builds the tree T corresponding to the optimal code in a bottom-up manner. It begins with the set of $|C|$ leaves and performs a sequence of $|C| - 1$ "merging" operation to create a final tree. The algorithm uses the minimum priority queue Q, keyed on the freq attribute, to identity the two least frequent objects to merge together. When we merge two objects, the result is a new object whose frequency is the sum of the frequencies of the two objects that were merged. The merging of the least frequency leaves continues till there tree is only left with 3 nodes, which will be the left leaf, right leaf and the sum of the left leaf and the right leaf as the root.

5 Solving the Problem:

Pseudocode for HUFFMAN (C):

```
1. $n = |C|$
2. $Q = C$
3. for $i = 1$ to $n - 1$
        allocate a new node z
        z.left = x = EXTRACT-MIN (Q)
        z.right = y = EXTRACT-MIN (Q)
        z.freq = x.freq + y.freq
4. INSERT (Q, z)
5. return EXTRACT-MIN(Q)
```

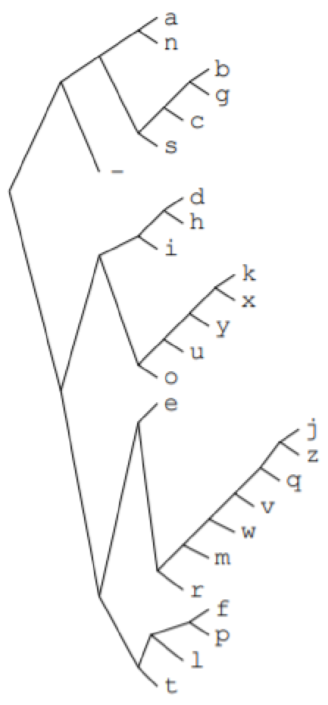
By solving the above pseudocode, we created the codewords for the corresponding input string and taking into account the probability/ frequency of the alphabets. We considered the following frequency data.



letter	code	length	frequency
A	0001	4	0.0817
B	101001	6	0.0145
C	11001	5	0.0248
D	00000	5	0.0431
E	100	3	0.1232
F	001100	6	0.0209
G	001101	6	0.0182
H	0101	4	0.0668
I	0100	4	0.0689
J	110100101	9	0.0010
K	1101000	7	0.0080
L	00001	5	0.0397
M	10101	5	0.0277
N	0110	4	0.0662
O	0010	4	0.0781
P	101000	6	0.0156
Q	1101001000	10	0.0009
R	1011	4	0.0572
S	0111	4	0.0628
T	111	3	0.0905
U	00111	5	0.0304
V	110101	6	0.0102
W	11000	5	0.0264
X	11010011	8	0.0015
Y	11011	5	0.0211
Z	1101001001	10	0.0005

(a) Courtesy – Discrete Mathematics and Its Application, By Kenneth. H. Rosen

*The Values of Frequency may change as per the internet source



(b) Courtesy – Information Theory,Inference, and learning Algorithm by David. C. Mackay

Thus, by implementing the above pseudocode we coded the program which takes the input as a string from the user and implements the above stated algorithm and the data of frequency and make convert each character into a codeword and displays it as a result. Further it appends all the codewords and generates an encoded message which is ready to be transmitted.

6 The significance of Discrete Mathematics

6.1 Time complexity:

If there are n nodes, EXTRACT-MIN () is called $2*(n–1)$ times. EXTRACT-MIN () takes $O(logn)$ time. Thus, a total time of $O(nlogn)$ where n is the number of unique characters.

6.2 B. Trees:

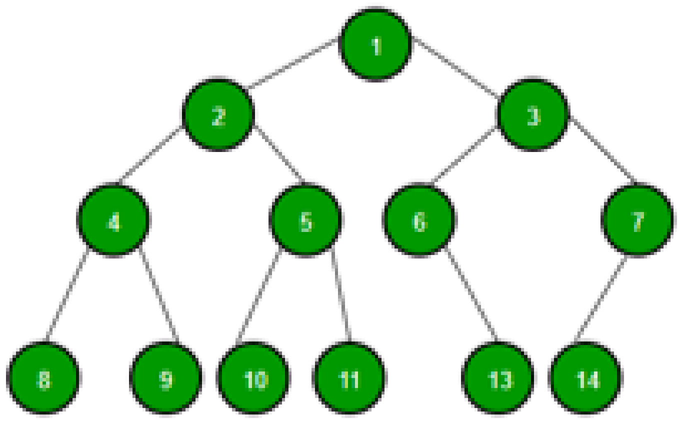
It constructs efficient algorithms for locating items in a list, even it helps determine the computational complexity of algorithms based on a sequence of decisions, such as sorting algorithms, thus is make it easy to transverse through the characters input sort the character with least frequency/ probability, even because of dynamic memory allocation, it uses less space in order to compute the algorithm and thus saves times and memory space.

7 Implementation and Significance of the application of the solution

Huffman codes compress data very effectively, saving of 20% to 90% are typical, depending on the characteristic of the data being compressed. It is currently one of the best algorithms for channel encoding process and there is no better algorithm.

8 MATHEMATICAL APPROACH

We have made use of Binary trees in order to assign codewords to characters, where with each new level in the tree we add 1 to the length of the codewords assigned.



Also in order to calculate how often each character is bound to come up we have calculated its probability.

Prefix free codes have been used (using them creates no ambiguity while decoding)

<u>Prefix free code:</u>
Codewords which fulfill the prefix free condition are called prefix codes.
Any one codeword of a system of codewords cannot be a prefix for a other codeword of the same system.
Example:

A: 00
B: 010
C: 110
D: 10

We can see that the codeword for A which is 00 is prefix to no other codeword in the list. This is called prefix free code and it helps decode the characters without any ambiguity.

<u>Why is prefix code used here?</u>
Basically it helps tackle ambiguity at the time of decoding. Lets understand tat with the help of an example: Say E was also an character in the list with the codeword 0001 then as soon as the compiler would read the birst two bits 00 it would think we are referring to the character A which causes errors at the time of decoding.

9 COMMERCIALIZATION SCOPE

- Huffman Coding technique is currently used over the world for encoding data as it comprises efficiently and thus reduces the amount of data transfer . Due to the lack of patent coverage, simplicity and speed, prefix codes are widely used. So it can be used to create a data compression service, that on the surface would be free but still make money through ads.

- Huffman coding is not used directly in real world but several derivatives of Huffman encoding algorithm is used in application like JPEG, PNG, MP3 etc. It can be implemented in combination with these compression techniques to compress variety of data like images, music, video files.

- It can also be used to create an encryption service, if we use as the key a probability distribution then we can create Huffman codes for each characters that can be used to encrypt them (But we will be trading off the compression for encryption).

Now where would such encryption finds it use? There is a pretty well knows way some people use to crack the key and that is 'Brute force' method, so if we Huffman encode a text before encrypting it, the attacker would not only has to decrypt the text but also Huffman decode it to get to the password.