Huffman, Shannon Encoding and Decoding

By:Sudhamshu B N(1DS18Ec091)
Rajasudhan Gowda S A(1DS19EC426)

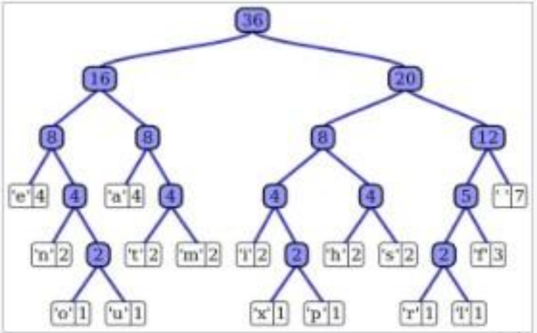
Submitted to:
Prof . Navya Holla

Huffman coding

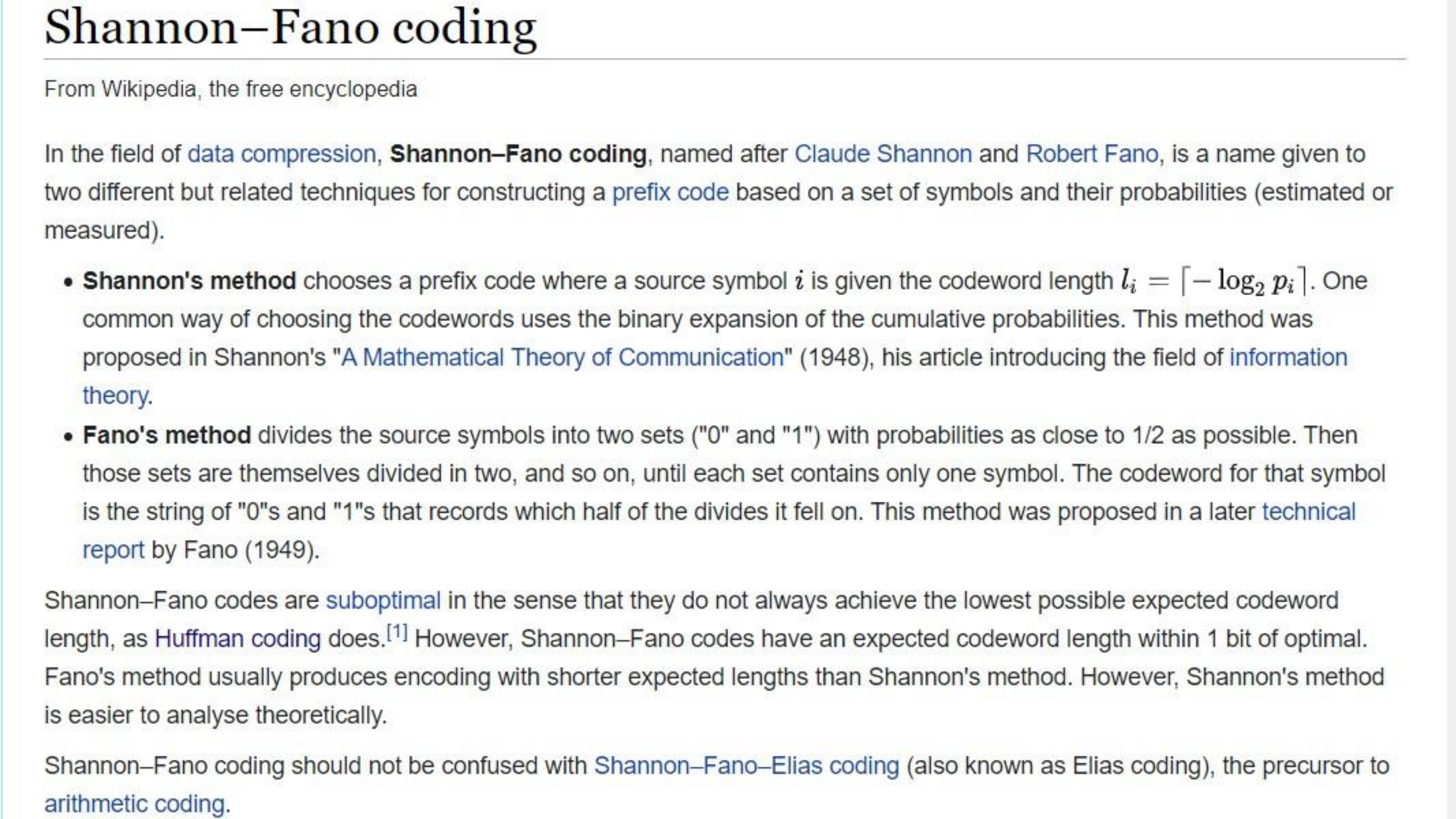
From Wikipedia, the free encyclopedia

In computer science and information theory, a **Huffman code** is a particular type of optimal prefix code that is commonly used for lossless data compression. The process of finding or using such a code proceeds by means of **Huffman coding**, an algorithm developed by David A. Huffman while he was a Sc.D. student at MIT, and published in the 1952 paper "A Method for the Construction of Minimum-Redundancy Codes".[1]

The output from Huffman's algorithm can be viewed as a variable-length code table for encoding a source symbol (such as a character in a file). The algorithm derives this table from the estimated probability or frequency of occurrence (*weight*) for each possible value of the source symbol. As in other entropy encoding methods, more common symbols are generally represented using fewer bits than less common symbols. Huffman's method can be efficiently implemented, finding a code in time linear to the number of input weights if these weights are sorted.[2] However, although optimal among methods encoding symbols separately, Huffman coding is not always optimal among all compression methods - it is replaced with arithmetic coding or asymmetric numeral systems if better compression ratio is required.



Huffman tree generated from the exact frequencies of the text "this is an example of a huffman tree". The frequencies and codes of each character are below. Encoding the sentence with this code requires 135 (or 147) bits, as opposed to 288 (or 180) bits if 36 characters of 8 (or 5) bits were used. (This assumes that the code tree structure is known to the decoder and thus does not need to be counted as part of the transmitted information.)



Shannon–Fano coding

From Wikipedia, the free encyclopedia

In the field of data compression, **Shannon–Fano coding**, named after Claude Shannon and Robert Fano, is a name given to two different but related techniques for constructing a prefix code based on a set of symbols and their probabilities (estimated or measured).

- **Shannon's method** chooses a prefix code where a source symbol i is given the codeword length $l_i = \lceil -\log_2 p_i \rceil$. One common way of choosing the codewords uses the binary expansion of the cumulative probabilities. This method was proposed in Shannon's "A Mathematical Theory of Communication" (1948), his article introducing the field of information theory.
- **Fano's method** divides the source symbols into two sets ("0" and "1") with probabilities as close to 1/2 as possible. Then those sets are themselves divided in two, and so on, until each set contains only one symbol. The codeword for that symbol is the string of "0"s and "1"s that records which half of the divides it fell on. This method was proposed in a later technical report by Fano (1949).

Shannon–Fano codes are suboptimal in the sense that they do not always achieve the lowest possible expected codeword length, as Huffman coding does.[1] However, Shannon–Fano codes have an expected codeword length within 1 bit of optimal. Fano's method usually produces encoding with shorter expected lengths than Shannon's method. However, Shannon's method is easier to analyse theoretically.

Shannon–Fano coding should not be confused with Shannon–Fano–Elias coding (also known as Elias coding), the precursor to arithmetic coding.

Algorithm-Huffman Encoding

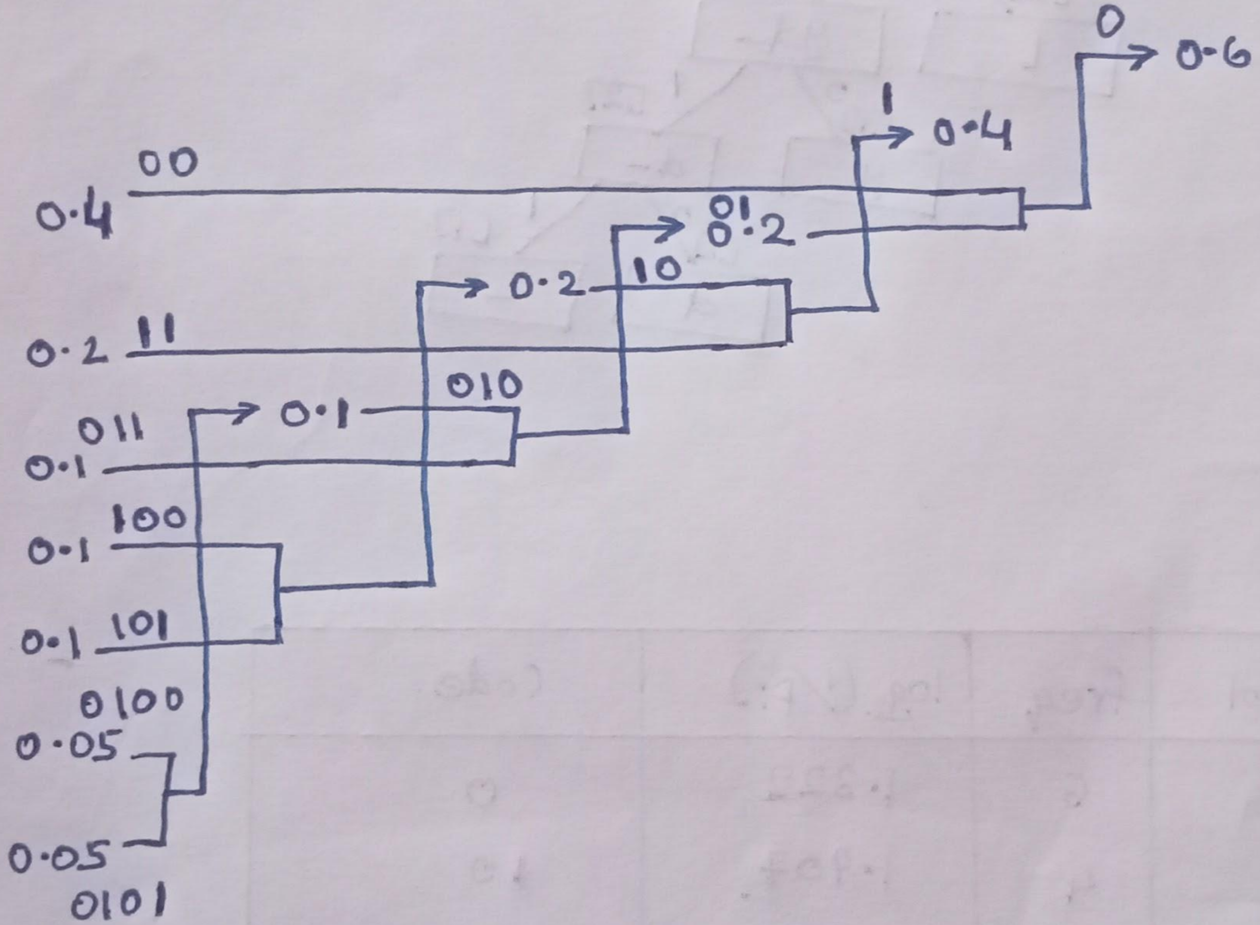
- Calculate the frequency of each character in the string.
- Sort the characters in increasing order of the frequency. These are stored in a priority queue
- Make each unique character as a leaf node
- Create an empty node. Assign the minimum frequency to the left child of the new node created and assign the second minimum frequency to the right child of the new node. Set the value of the new node as the sum of the above two minimum frequencies.
- Remove these two minimum frequencies from the queue and add the sum into the list of frequencies.
- Insert node new node into the tree.
- Repeat steps 3 to 5 for all the characters
- For each non-leaf node, assign 1 to the left edge and 0 to the right edge.

Algorithm-Huffman Decoding

- To decode the encoded data we require Huffman tree and compressed code.

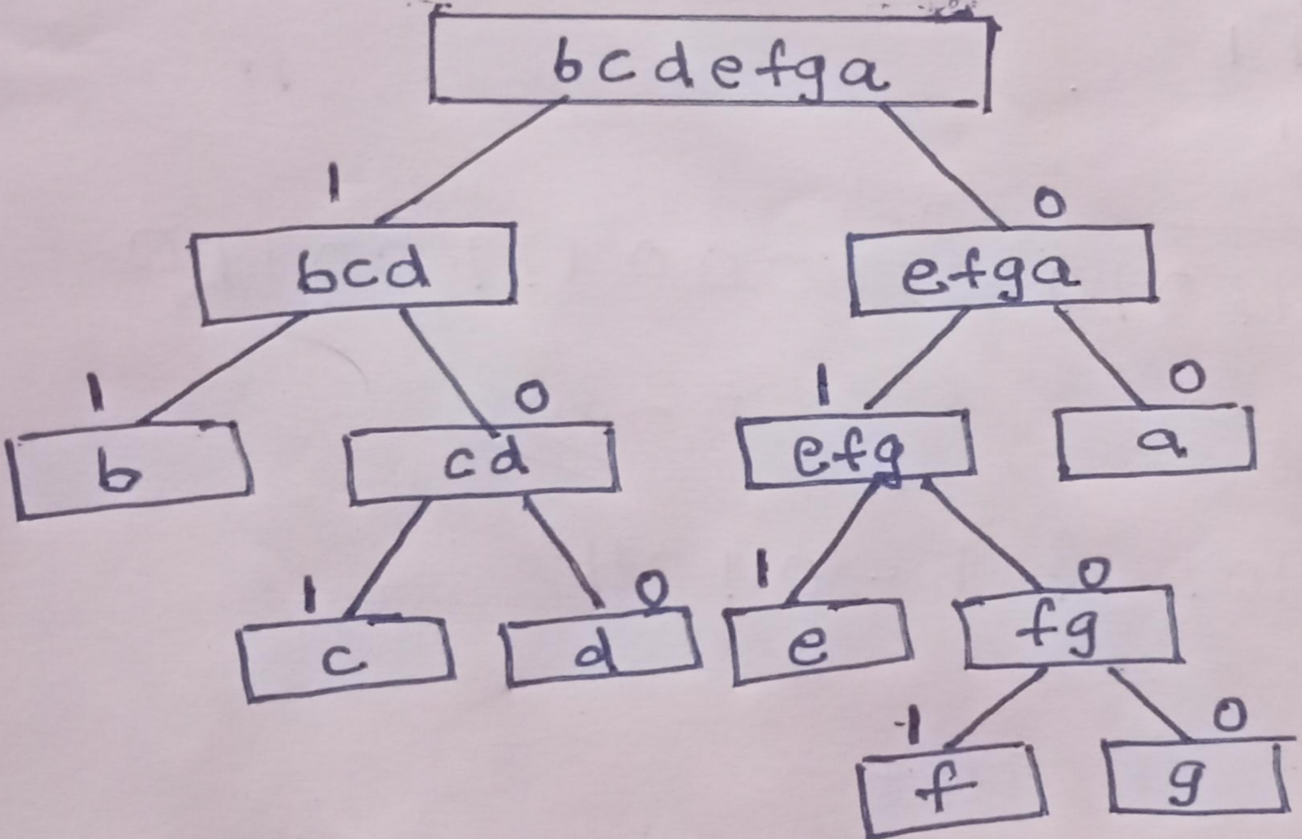
- Iterate through the binary encoded data,to find the corresponding encoded data.

Input : aaaaaaabbbbccddeefg



Probability values [0.4, 0.2, 0.1, 0.1, 0.1, 0.05, 0.05]

Input: aaaaaaaabbbbccddeefg



Algorithm-Shannon Encoding

- Calculate the frequency of each character in the string.
- Sort the characters in increasing order of the frequency. These are stored in a priority queue
- Recursively divide the symbols into two parts, each with approximately the same number of counts, until all parts contain only one symbol.
- Assign values to the nodes (at the left of branch 0 and at the right 1)

Algorithm-Shannon Decoding

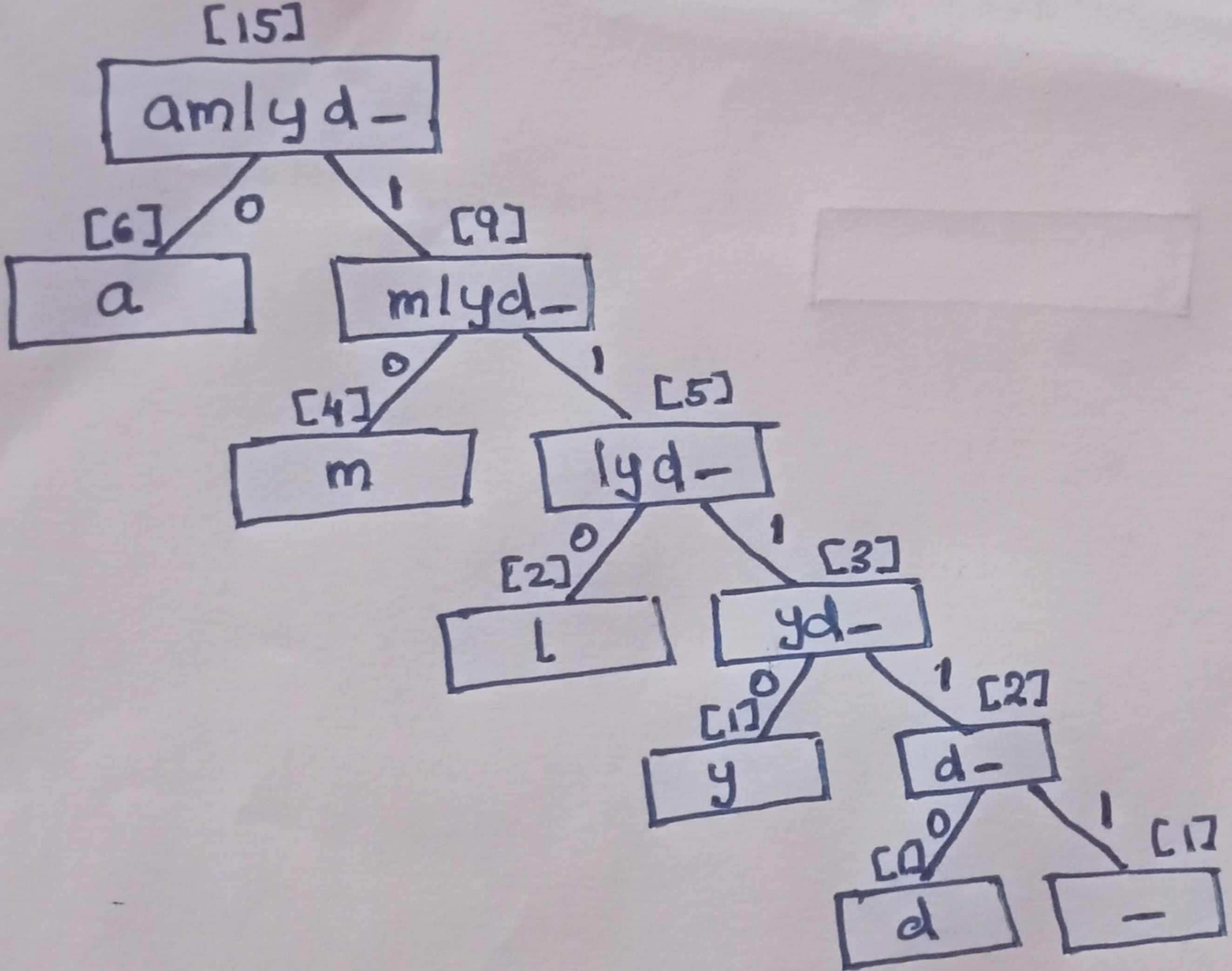
- To decode the data the same code as Huffman decoding is used.

- To decode the encoded data we require Shannon tree and compressed code.

- Iterate through the binary encoded data,to find the corresponding encoded data.

malayalam madam

Symbol	freq	$\log_2(1/p_i)$	Code.
a	6	1.322	0
m	4	1.907	10
l	2	2.907	110
y	1	3.907	1110
d	1	3.907	11110
–	1	3.907	11111


[15]
amlyd_
0
1
[6]
a
[9]
mlyd_
0
1
[4]
m
[5]
lyd_
0
1
[2]
l
[3]
yd_
0
1
[1]
y
[2]
d_
0
1
[1]
d
[1]
_

Results-Huffman Encoding and Decoding

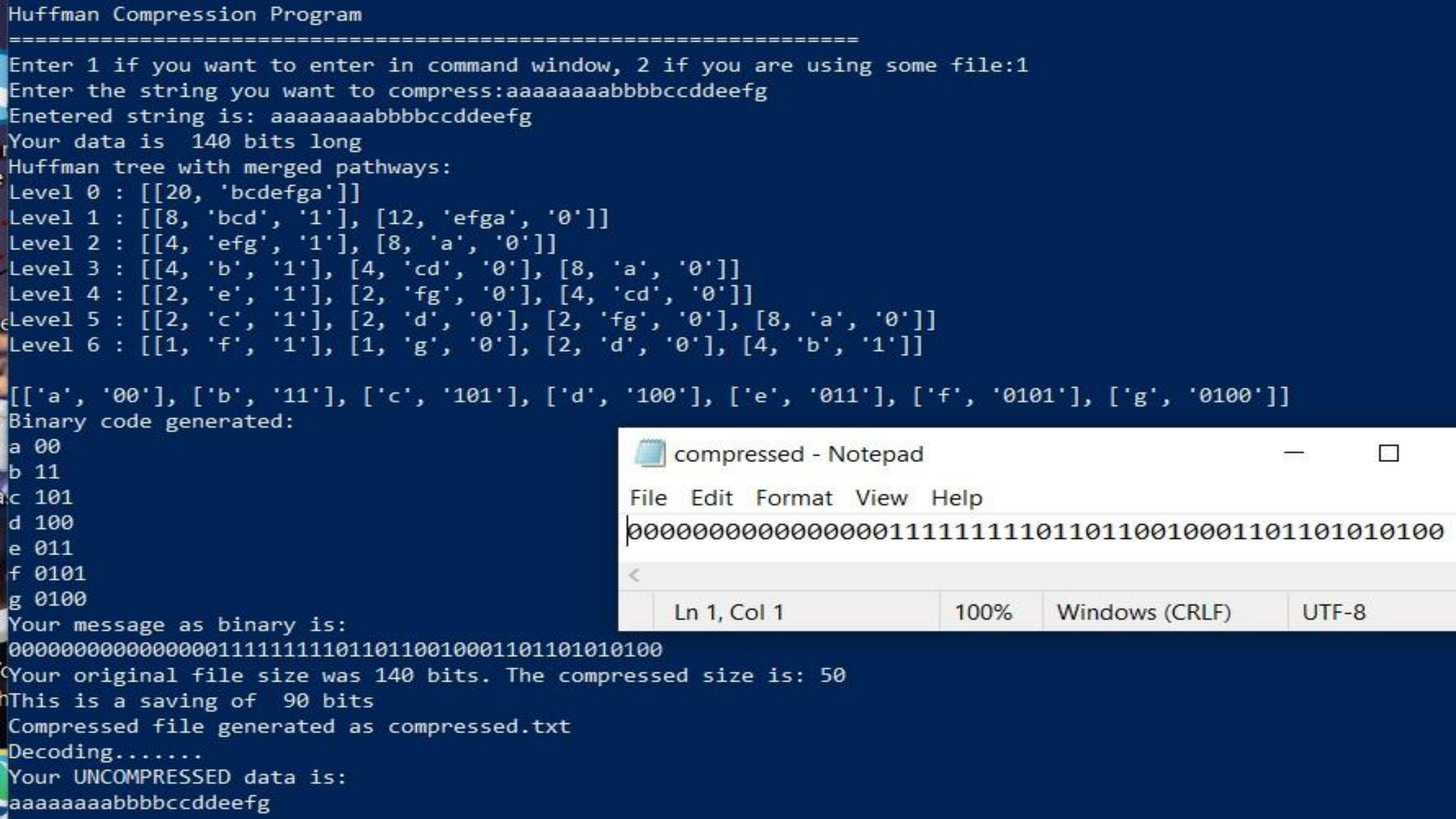
Executed Huffman Encoding and decoding program for probability values

[0.4, 0.2, 0.1, 0.1, 0.1, 0.05, 0.05] generated codes 00,11,101,100,011,0101,0100
[8*a,4*b,2*c,2*d,2*e, 1*f,   1*g]

compressed file generated as :

0000000000000000111111110110110010001101101010100

Decoded data:aaaaaaaabbbbccddeefg



```
Huffman Compression Program
=================================================================
Enter 1 if you want to enter in command window, 2 if you are using some file:1
Enter the string you want to compress:aaaaaaaabbbbccddeefg
Enetered string is: aaaaaaaabbbbccddeefg
Your data is  140 bits long
Huffman tree with merged pathways:
Level 0 : [[20, 'bcdefga']]
Level 1 : [[8, 'bcd', '1'], [12, 'efga', '0']]
Level 2 : [[4, 'efg', '1'], [8, 'a', '0']]
Level 3 : [[4, 'b', '1'], [4, 'cd', '0'], [8, 'a', '0']]
Level 4 : [[2, 'e', '1'], [2, 'fg', '0'], [4, 'cd', '0']]
Level 5 : [[2, 'c', '1'], [2, 'd', '0'], [2, 'fg', '0'], [8, 'a', '0']]
Level 6 : [[1, 'f', '1'], [1, 'g', '0'], [2, 'd', '0'], [4, 'b', '1']]

[['a', '00'], ['b', '11'], ['c', '101'], ['d', '100'], ['e', '011'], ['f', '0101'], ['g', '0100']]
Binary code generated:
a 00
b 11
c 101
d 100
e 011
f 0101
g 0100
Your message as binary is:
00000000000000001111111110110110010001101101010100
Your original file size was 140 bits. The compressed size is: 50
This is a saving of  90 bits
Compressed file generated as compressed.txt
Decoding.......
Your UNCOMPRESSED data is:
aaaaaaaabbbbccddeefg
```

compressed - Notepad

```
00000000000000001111111110110110010001101101010100
```

Results-Shannon Encoding and Decoding

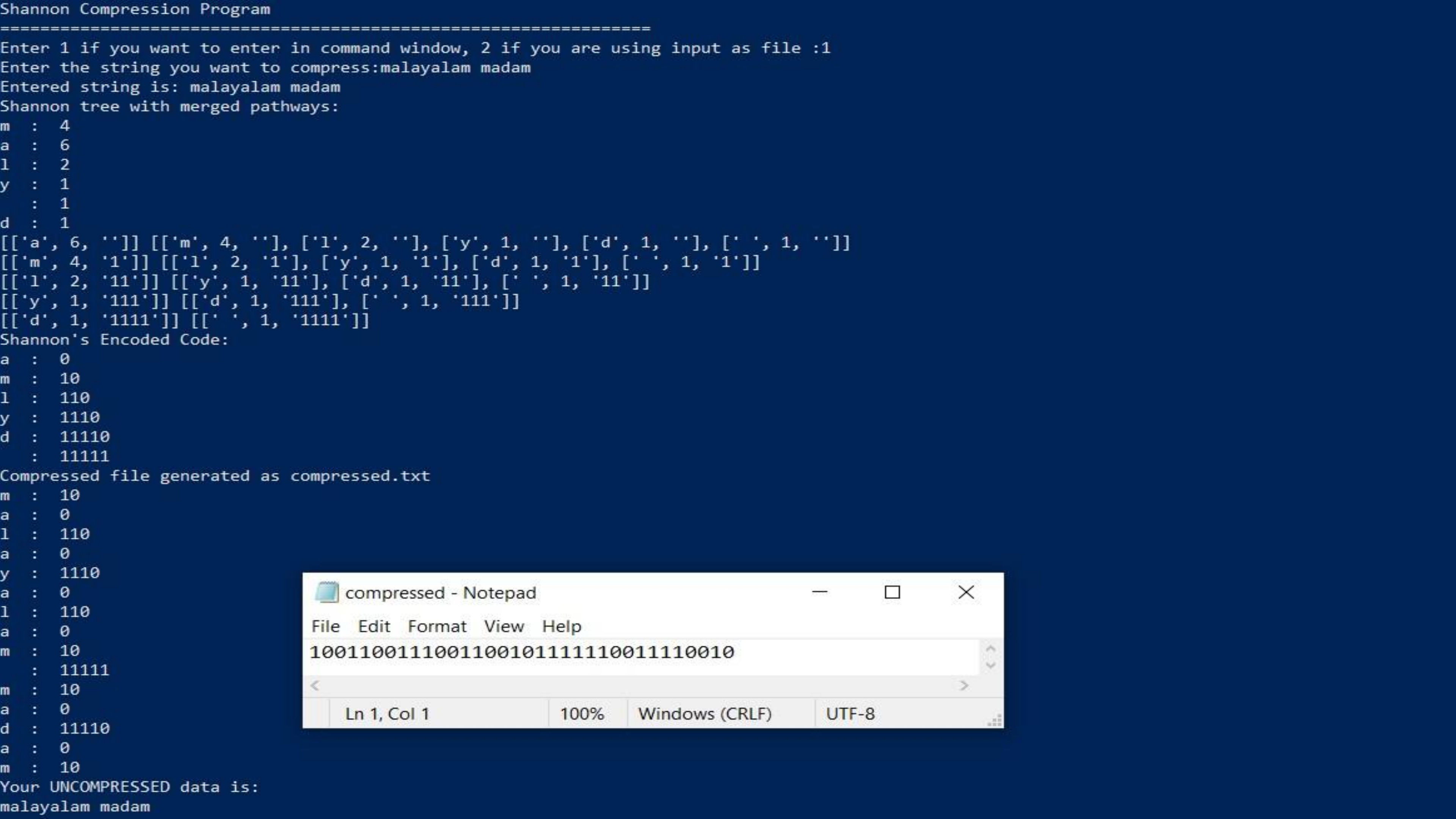
Executed Shannon Encoding and decoding for input “malayalam madam”

for probability values [0.266, 0.4, 0.1333, 0.0666,0.0666,0.0666]
[a , m , l , y , d , “ “]

Generated Codes: 0,10,110,1110,11110,11111

Binary code :1001100111001100101111110011110010

Decoded data: malayalam madam



```
Shannon Compression Program
=================================================================
Enter 1 if you want to enter in command window, 2 if you are using input as file :1
Enter the string you want to compress:malayalam madam
Entered string is: malayalam madam
Shannon tree with merged pathways:
m  :  4
a  :  6
l  :  2
y  :  1
   :  1
d  :  1
[['a', 6, '']] [['m', 4, ''], ['l', 2, ''], ['y', 1, ''], ['d', 1, ''], [' ', 1, '']]
[['m', 4, '1']] [['l', 2, '1'], ['y', 1, '1'], ['d', 1, '1'], [' ', 1, '1']]
[['l', 2, '11']] [['y', 1, '11'], ['d', 1, '11'], [' ', 1, '11']]
[['y', 1, '111']] [['d', 1, '111'], [' ', 1, '111']]
[['d', 1, '1111']] [[' ', 1, '1111']]
Shannon's Encoded Code:
a  :  0
m  :  10
l  :  110
y  :  1110
d  :  11110
   :  11111
Compressed file generated as compressed.txt
m  :  10
a  :  0
l  :  110
a  :  0
y  :  1110
a  :  0
l  :  110
a  :  0
m  :  10
   :  11111
m  :  10
a  :  0
d  :  11110
a  :  0
m  :  10
Your UNCOMPRESSED data is:
malayalam madam
```

compressed - Notepad

```
1001100111001100101111110011110010
```

Mini project Files

sudhamshu091/Huffman-Encoding-and-Decoding (github.com)

sudhamshu091/Shannon-Encoding-and-Decoding (github.com)

DCS-Assignment- Report Template

Discussions

Huffman encoding is more advantageous than Shannon encoding for the following reasons.

- Greedy algorithm
- Minimizes the length of code generated depending on frequency
- No prefix binary code

Applications-Huffman Encoding

- Huffman encoding is widely used in compression formats like GZIP, PKZIP (winzip) and BZIP2.
- Multimedia codecs like JPEG, PNG and MP3 use Huffman encoding.
- Huffman encoding still dominates the compression industry since newer arithmetic and range coding schemes are avoided due to their patent issues.
- Brotli compression algorithm by Google compresses data using a combination of a modern variant of the LZ77 algorithm, Huffman coding and 2nd order context modeling.

Thank You