

# LZ77,Huffman,Shannon Encoding and Decoding for text input, LZ77,Huffman,Shannon Encoding,decoding for Image Compression

By:Sudhamshu B N  
Rajasudhan Gowda S A

Submitted to:  
Prof . Deepa N P

# LZ77 Encoding-Algorithm

- Set the coding position to the beginning of the input stream.
- Find the longest match in the window for the lookahead buffer.
- If a match is found, output the pointer P. Move the coding position (and the window) L bytes forward.
- If a match is not found, output a null pointer and the first byte in the lookahead buffer. Move the coding position (and the window) one byte forward.
- If the lookahead buffer is not empty, return to step 2.

# Example

Input: “cabracadabrarrarrad”

Window Size: 13

Search Window Size: 7

Preview/Lookup Window Size: 6



# LZ77 Encoding

$\langle 0, 0, c \rangle$

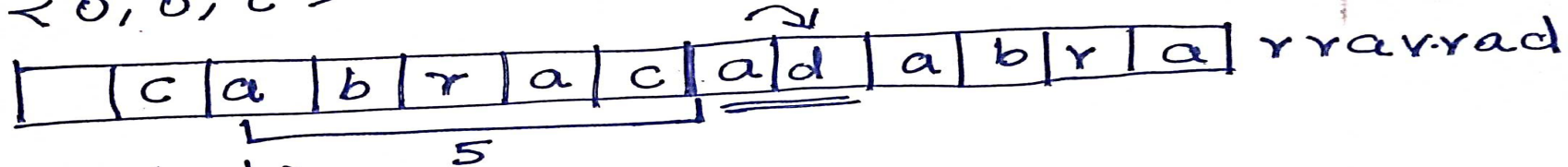
$\langle 0, 0, a \rangle$

$\langle 0, 0, b \rangle$

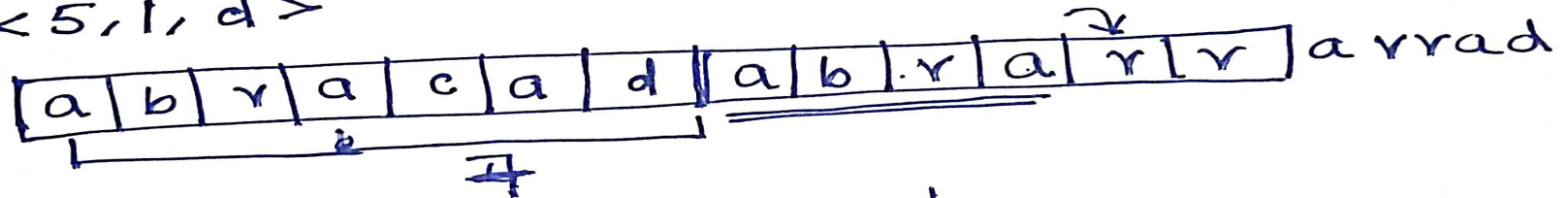
$\langle 0, 0, r \rangle$

$\langle 0, 0, a \rangle$

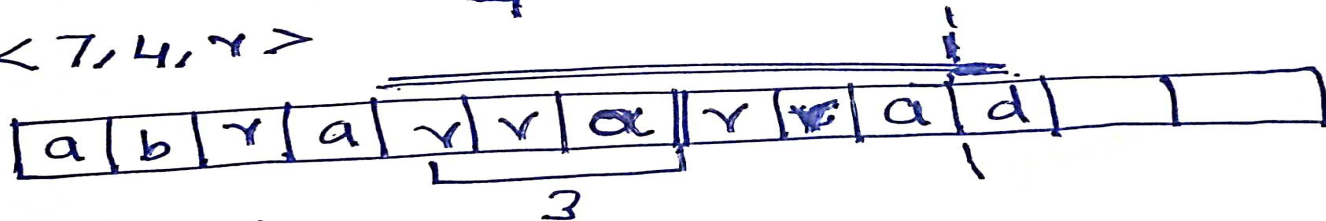
$\langle 0, 0, c \rangle$



$\langle 5, 1, d \rangle$



$\langle 7, 4, r \rangle$



$\langle 3, 5, d \rangle$

# LZ77 Decoding

$\langle 0, 0, c \rangle$

$\langle 0, 0, a \rangle$

$\langle 0, 0, b \rangle$

$\langle 0, 0, r \rangle$

$\langle 0, 0, a \rangle$

$\langle 0, 0, c \rangle$

$\langle 5, 1, d \rangle$

$\langle 7, 4, r \rangle$

$\langle 3, 5, d \rangle$

					c
				c	a
			c	a	b
		c	a	b	r
	c	a	b	r	a
c	a	b	r	a	c

5

c	a	b	r	a	c	a	d
---	---	---	---	---	---	---	---

7

c	a	b	r	a	c	a	d	a	b	r	a	r
---	---	---	---	---	---	---	---	---	---	---	---	---

3

c	a	b	r	a	c	a	d	a	b	r	a	r	r	a	r	a	d
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---



# Algorithm-Huffman Encoding

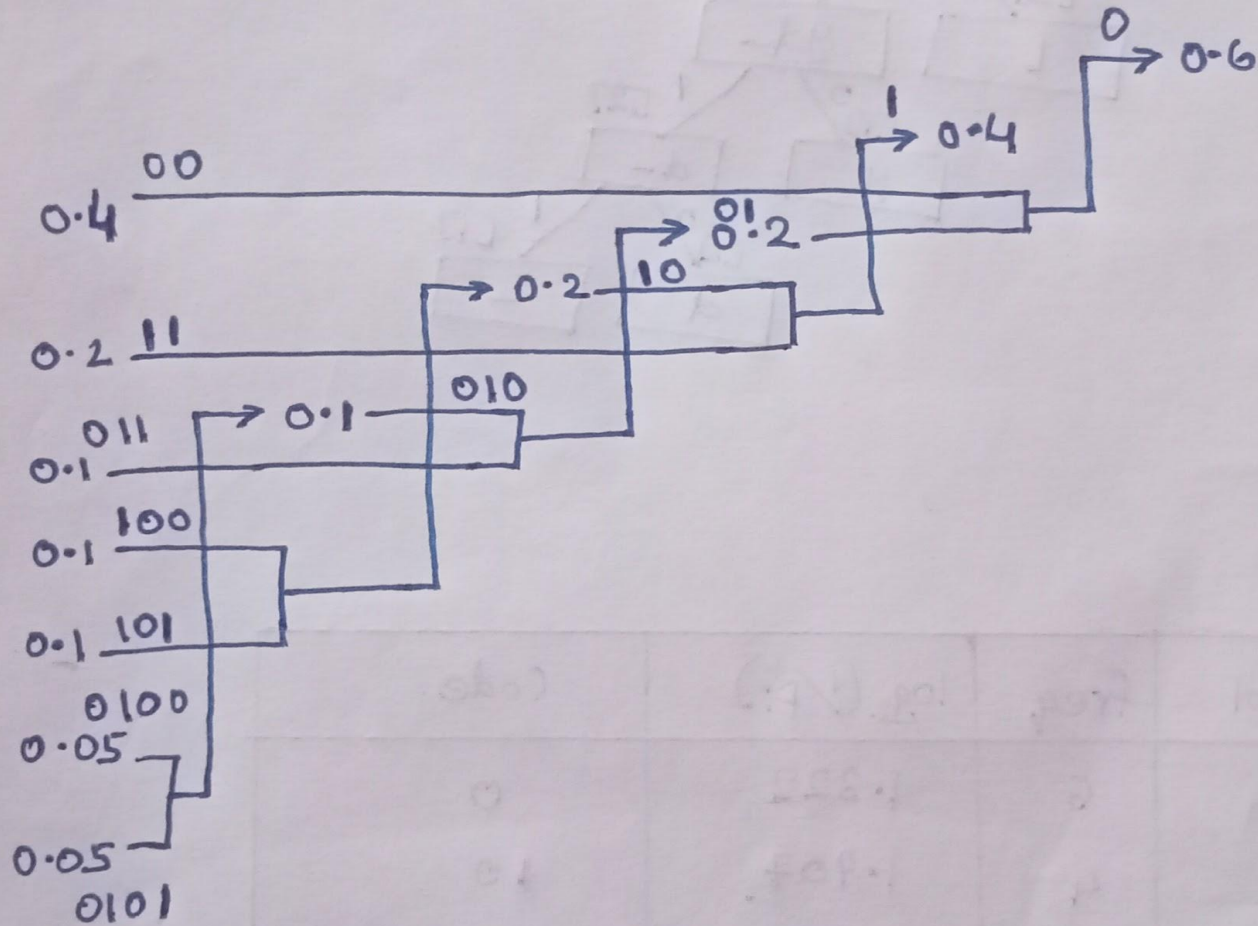
- Calculate the frequency of each character in the string.
- Sort the characters in increasing order of the frequency. These are stored in a priority queue
- Make each unique character as a leaf node
- Create an empty node. Assign the minimum frequency to the left child of the new node created and assign 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.

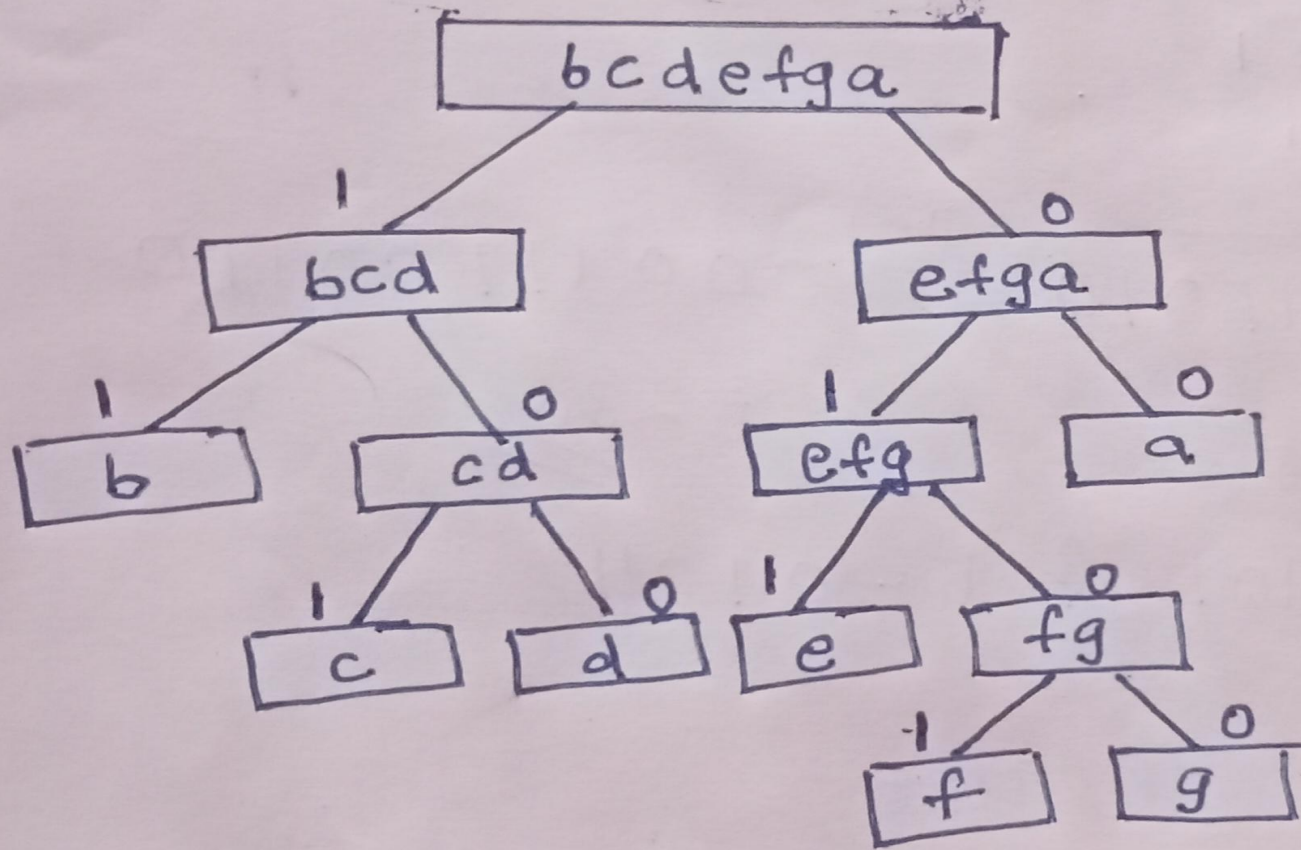
Input : aaaaaaabbbbbccddeefg



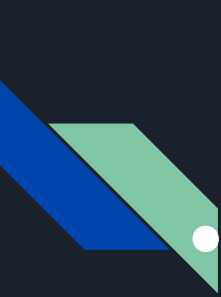


Probability values  $[0.4, 0.2, 0.1, 0.1, 0.1, 0.05, 0.05]$

Input: aaaaaaaabbbbbcddceefg



# 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)

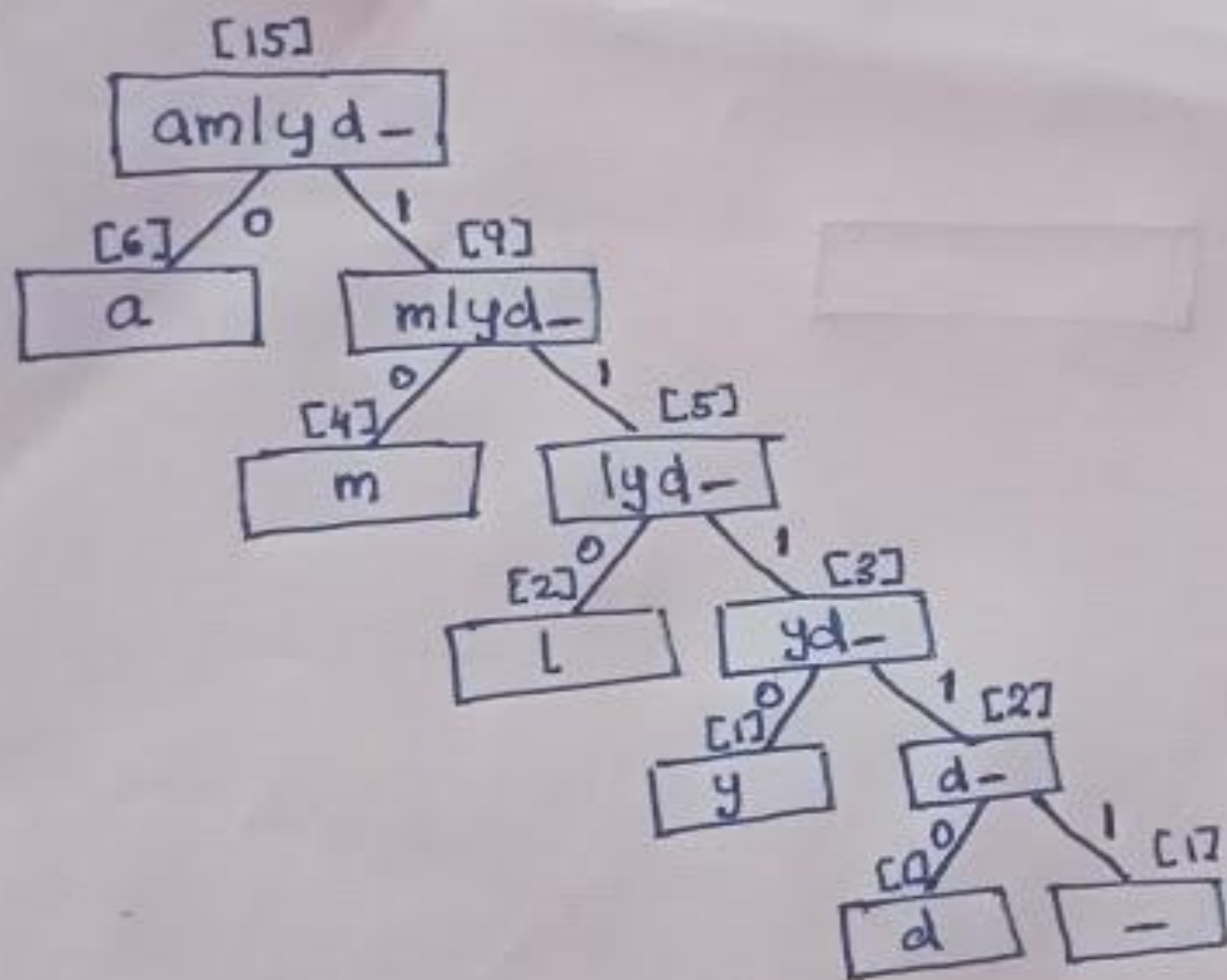


# Algorithm-Shannon Decoding

- 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



# Algorithm-Image Compression

- Convert the image file to numpy array values.
- Convert numpy array to list and list to a string.
- Perform the Compression algorithms and get the uncompressed string back.
- Use regex to convert string with brackets '[' ]' to only probability values with same dimension of input image.
- Convert the string values to numpy array values.
- Convert numpy array to image file.



# Mini project Files

[sudhamshu091/LZ77-Encoding-and-Decoding \(github.com\)](#)

[Huffman-Encoding-and-Decoding \(github.com\)](#)

[Shannon-Encoding-and-Decoding \(github.com\)](#)

[sudhamshu091/LZ77-Encoding-and-Decoding-for-Image-Compression \(github.com\)](#)

[sudhamshu091/Huffman-Encoding-Decoding-For-Image-Compression \(github.com\)](#)

[Shannon-Fano-Encoding-and-Decoding-for-Image-Compression \(github.com\)](#)

# Results-LZ77 Encoding and Decoding

## LZ77 Compression Algorithm

=====

Enter 1 if you want to enter input in command window, 2 if you are using some file:1

Enter the string you want to compress:cabracadabrarrarrad

Entered string is: cabracadabrarrarrad

Enter the Search Window Size:7

Enter the Preview Window Size:6

Compressed file generated as compressed.txt

Encoded string: { 0 : 0 : c } { 0 : 0 : a } { 0 : 0 : b } { 0 : 0 : r } { 0 : 0 : a } { 0 : 0 : c } { 5 : 1 : d }  
{ 7 : 4 : r } { 3 : 5 : d }

Decoded string: cabracadabrarrarrad

compressed - Notepad

File Edit Format View Help

```
[[0, 0, 0, 0, 0, 0, 5, 7, 3], [0, 0, 0, 0, 0, 0, 1, 4, 5],  
['c', 'a', 'b', 'r', 'a', 'c', 'd', 'r', 'd']]
```





# Results-Huffman Encoding and Decoding

Executed Huffman Encoding and decoding program for the following input: "Python programming is fun."

```
[[['P', '01110'], ['y', '1100'], ['t', '1110'], ['h', '01001'], ['o', '0001'], ['n', '100'], [' ', '101'],  
  ['p', '01000'], ['r', '0000'], ['g', '0101'], ['a', '01101'], ['m', '0010'], ['i', '0011'], ['s', '1111'], ['f',  
  '01100'], ['u', '1101'], [',', '01111']]]
```

compressed file generated as:

```
0111011001110010010001100101010000000000101010000011010010001000111000101  
101001111110101100110110001111
```

Decoded data:Python programming is fun.

# 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:aaaaaaaaabbbbccddeefg

Entered string is: aaaaaaaaaabbbbccddeefg

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:

0000000000000000111111110110110010001101101010100

Your original file size was 140 bits. The compressed size is: 50

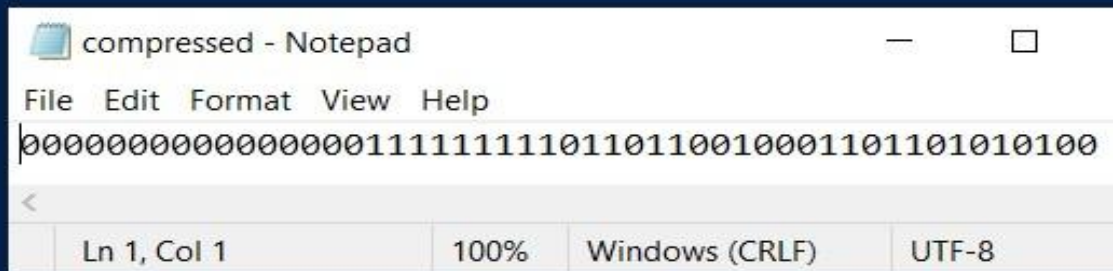
This is a saving of 90 bits

Compressed file generated as compressed.txt

Decoding.....

Your UNCOMPRESSED data is:

aaaaaaaaabbbbccddeefg





## Results

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

# Results-Shannon Encoding and Decoding

## 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

d : 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

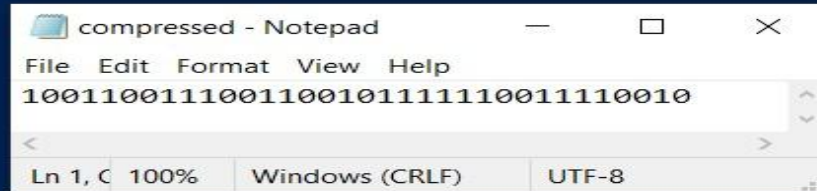
d : 11110

a : 0

m : 10

Your UNCOMPRESSED data is:

malayalam madam



# Image Compression Results

Input Image



Output Image



Input



Output







Input



Output





# Discussions

- LZ77 is much better compared to Huffman and Shannon in terms of speed
- Huffman is advantageous Compared to Shannon's Algorithm
- The Image Compression algorithm used by us is not the most efficient one, but we have just tried to compress image files.
- LZ77 has proven better results for image compression
- LZ78 algorithm is also a great method for data compression





# Applications

- 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.
- Image Compression



Thank You