Huffman Coding

MD. MUKTAR HOSSAIN

LECTURER

DEPT. OF CSE

VARENDRA UNIVERSITY

Data Compression

Data compression is the process of reducing the size of data to save storage space or transmission time.

Types of Data Compression

- Lossless Compression
    - Retains the original data exactly after decompression.
    - Used for text files, program files, and medical imaging.
    - Examples: Huffman Coding, Run-Length Encoding
- Lossy Compression
    - Some data is lost, but the reduction in size is significant.
    - Used for images, audio, and videos where perfect accuracy isn't needed.
    - Examples: JPEG, MP3, MPEG, H.264, H.265

Character Encoding Compression

Encoding characters is a key part of lossless compression algorithms, where data is compressed without losing any information, and the original data can be perfectly reconstructed during decompression.

Types of Character Encoding

- Fixed-Length Encoding
  - Each character or symbol is represented using a fixed number of bits.
- Variable-Length Encoding
  - Different characters or symbols are represented using a varying number of bits.
  - More frequent symbols get shorter codes, while less frequent ones get longer codes.

Fixed-Length Encoding

M = “Sleeplessness stresses restless senses endlessly”

s	e	l	sp	n	t	r	p	d	y
17	12	5	4	3	2	2	1	1	1

Total number of characters = 48

ASCII

- Uses 7-bit codes to represent characters (A-Z, a-z, 0-9, symbols).
- Required bits to encode **M** = 48*7 = 336 bits

Minimal Fixed Code

- Since there is 10 different character, we can use 4 bits for each character to encode message M.
- Required bits to encode **M** = 48*4 = 192 bits

Variable-Length Encoding

M = “Sleeplessness stresses restless senses endlessly”

s	e	l	sp	n	t	r	p	d	y
17	12	5	4	3	2	2	1	1	1

Total number of characters = 48

Huffman Coding can be used for variable length encoding.

Huffman Coding - Introduction

- Huffman coding is a greedy algorithm used for lossless data compression.
- It assigns variable-length binary codes to input characters. [shorter codes assigned to more frequent characters and longer codes assigned to less frequent ones]
- This minimizes the overall length of the encoded message.
- The core idea is to build a binary tree (Huffman tree) where each leaf node represents a character and its frequency. The tree is then traversed to assign binary codes.

Huffman Coding - Applications

- **File Storage –** Reduces disk space usage.
- **Network Transmission –** Speeds up data transfer.
- **Multimedia Streaming –** Reduces bandwidth requirements.
- **Cloud Storage –** Saves costs for large-scale data storage.
- **Data Backup & Archiving –** Efficiently stores historical data.

Huffman Coding - Algorithm

- Count the frequency of each character in the input.
- Insert all characters and their frequencies into a priority queue (min-heap).
- While there is more than one node in the queue:
    - Remove the two nodes with the lowest frequency.
    - Create a new internal node with these two as children and a frequency equal to their sum.
    - Insert the new node back into the priority queue.
- The last remaining node is the root of the Huffman tree.
- Generate binary codes by traversing the Huffman tree.

Huffman Coding - Example

M = “Sleeplessness stresses restless senses endlessly”

Count the frequency of each character in the input.

s	l	e	p	n	sp	t	r	d	y
17	5	12	1	3	4	2	2	1	1

Insert all characters and their frequencies into a priority queue (min-heap).

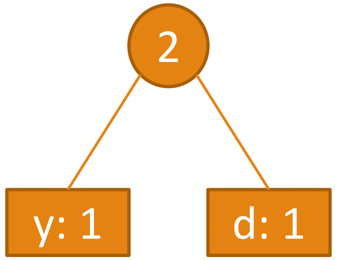
y: 1	d: 1	p: 1	r: 2	t: 2	n: 3	sp: 4	l: 5	e: 12	s: 17

Huffman Coding - Example

y: 1	d: 1	p: 1	r: 2	t: 2	n: 3	sp: 4	l: 5	e: 12	s: 17

- While there is more than one node in the queue:
  - Remove the two nodes with the lowest frequency.
  - Create a new internal node with these two as children and a frequency equal to their sum.
  - Insert the new node back into the priority queue.

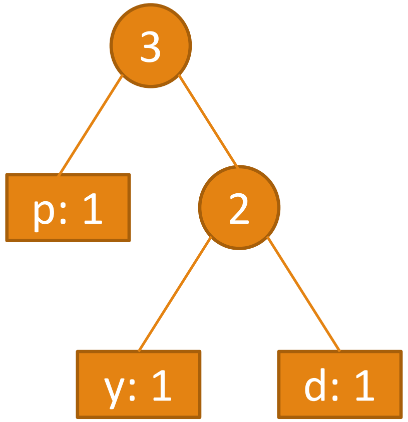
p: 1	2	r: 2	t: 2	n: 3	sp: 4	l: 5	e: 12	s: 17



Huffman Coding - Example

p: 1	2	r: 2	t: 2	n: 3	sp: 4	l: 5	e: 12	s: 17

- While there is more than one node in the queue:
  - Remove the two nodes with the lowest frequency.
  - Create a new internal node with these two as children and a frequency equal to their sum.
  - Insert the new node back into the priority queue.

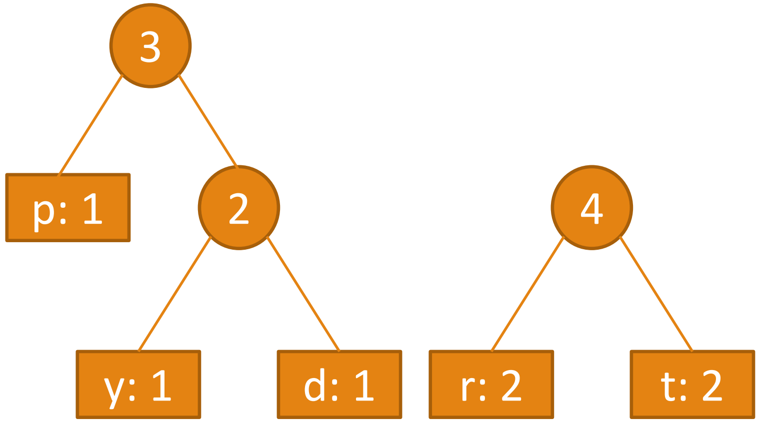


r: 2	t: 2	3	n: 3	sp: 4	l: 5	e: 12	s: 17

Huffman Coding - Example

r: 2	t: 2	3	n: 3	sp: 4	l: 5	e: 12	s: 17

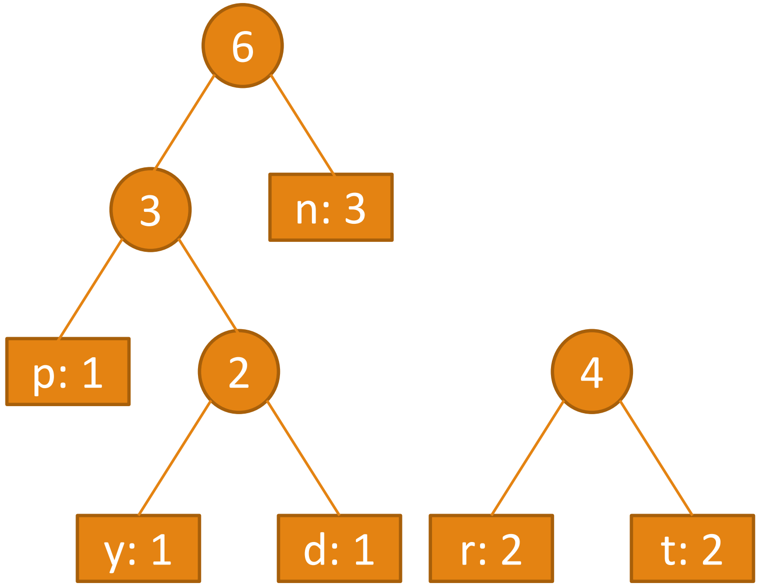
- While there is more than one node in the queue:
  - Remove the two nodes with the lowest frequency.
  - Create a new internal node with these two as children and a frequency equal to their sum.
  - Insert the new node back into the priority queue.



3	n: 3	4	sp: 4	l: 5	e: 12	s: 17

Huffman Coding - Example

3	n: 3	4	sp: 4	l: 5	e: 12	s: 17

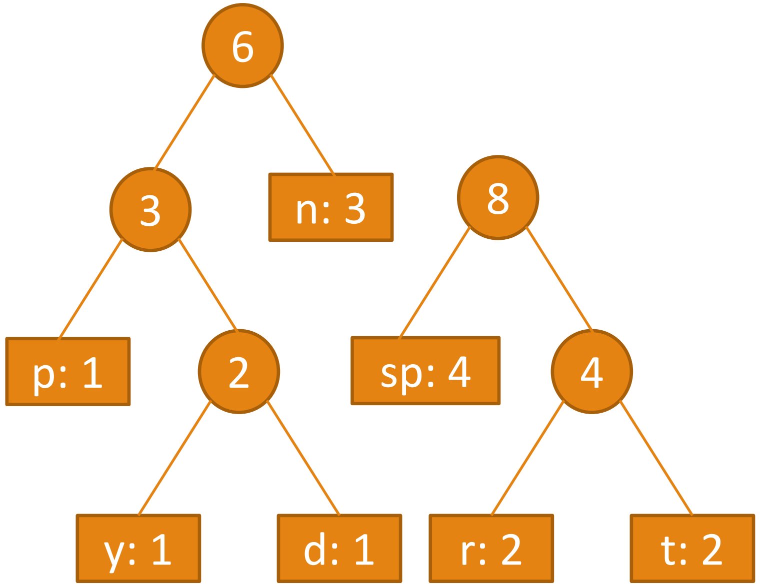


- While there is more than one node in the queue:
  - Remove the two nodes with the lowest frequency.
  - Create a new internal node with these two as children and a frequency equal to their sum.
  - Insert the new node back into the priority queue.

4	sp: 4	l:5	6	e: 12	s: 17

Huffman Coding - Example

4	sp: 4	l:5	6	e: 12	s: 17



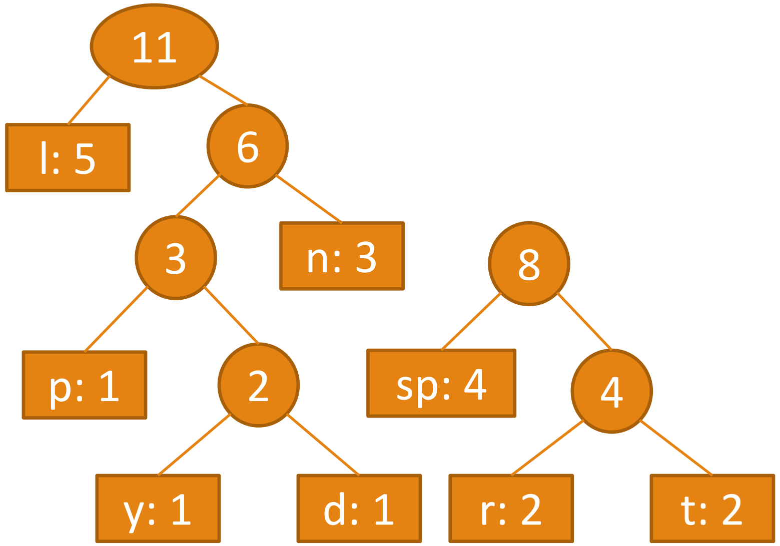
- While there is more than one node in the queue:
  - Remove the two nodes with the lowest frequency.
  - Create a new internal node with these two as children and a frequency equal to their sum.
  - Insert the new node back into the priority queue.

l: 5	6	8	e: 12	s: 17

Huffman Coding - Example

l: 5	**6**	**8**	**e: 12**	**s: 17**

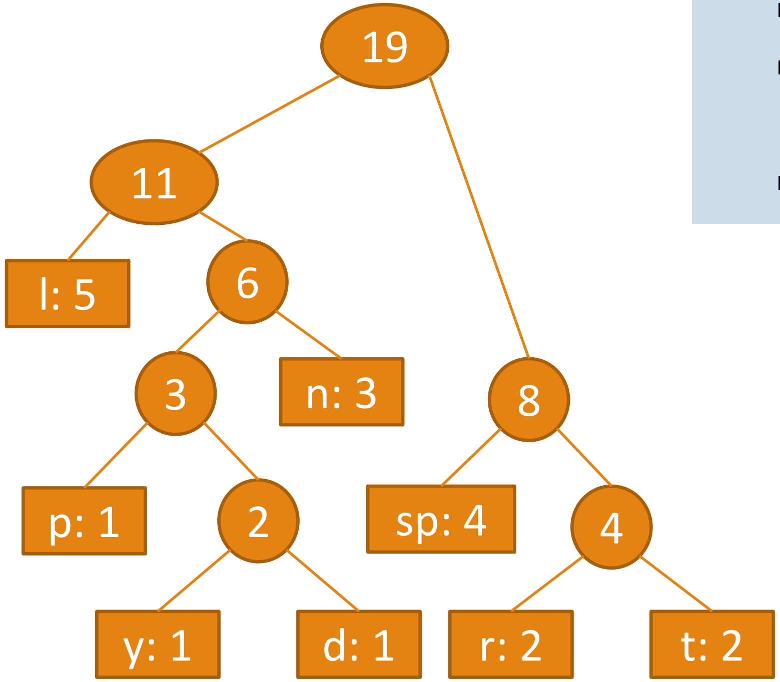
- While there is more than one node in the queue:
  - Remove the two nodes with the lowest frequency.
  - Create a new internal node with these two as children and a frequency equal to their sum.
  - Insert the new node back into the priority queue.



8	**11**	**e: 12**	**s: 17**

Huffman Coding - Example

8	**11**	**e: 12**	**s: 17**

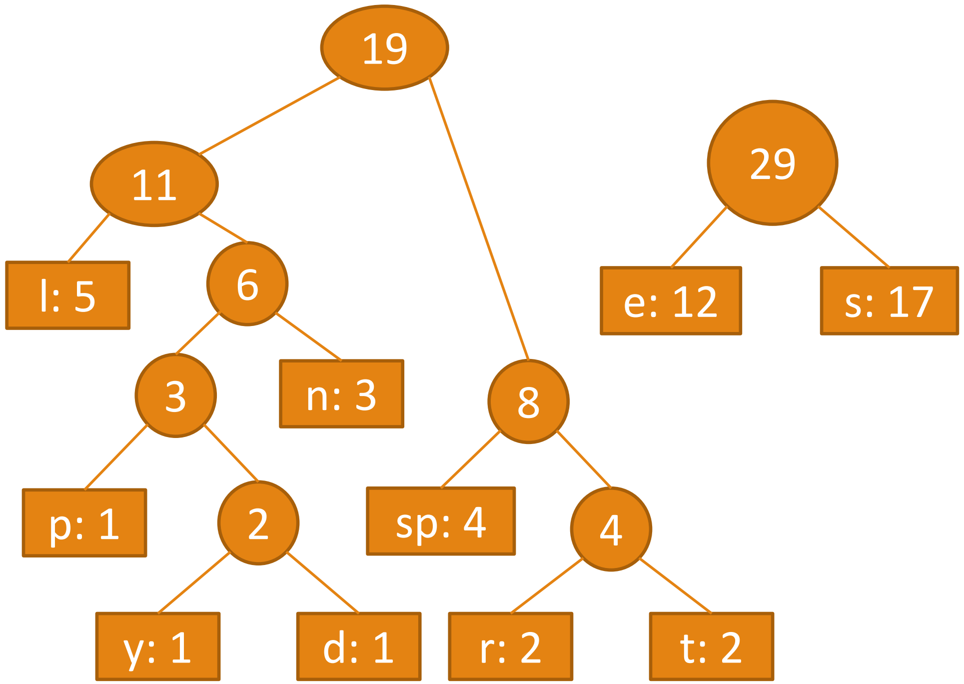


- While there is more than one node in the queue:
  - Remove the two nodes with the lowest frequency.
  - Create a new internal node with these two as children and a frequency equal to their sum.
  - Insert the new node back into the priority queue.

e: 12	**s: 17**	**19**

Huffman Coding - Example

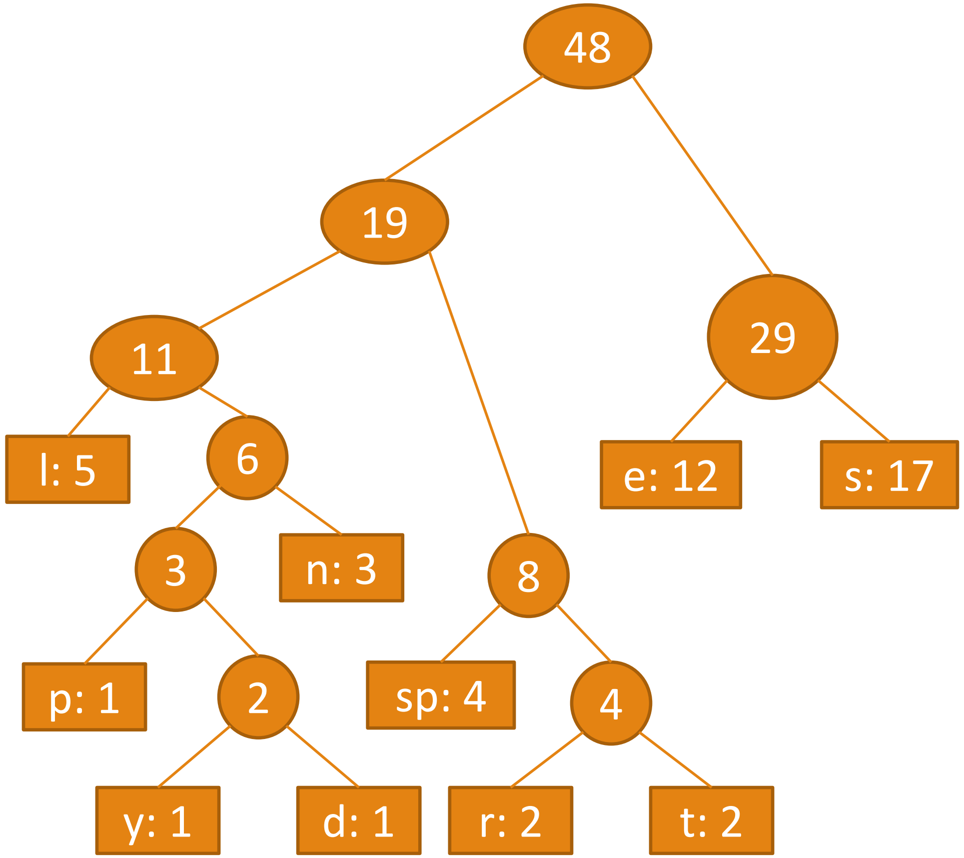
e: 12	**s: 17**	**19**



- While there is more than one node in the queue:
  - Remove the two nodes with the lowest frequency.
  - Create a new internal node with these two as children and a frequency equal to their sum.
  - Insert the new node back into the priority queue.

19	**29**

Huffman Coding - Example



19	**29**

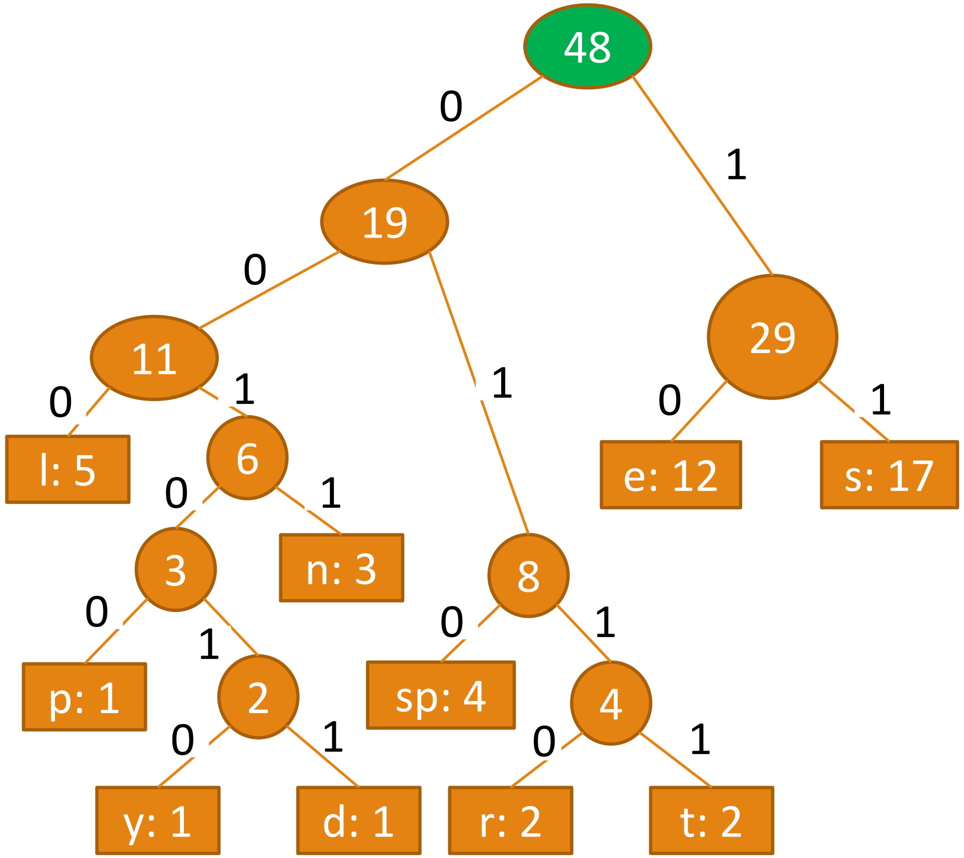
- While there is more than one node in the queue:
  - Remove the two nodes with the lowest frequency.
  - Create a new internal node with these two as children and a frequency equal to their sum.
  - Insert the new node back into the priority queue.

48

The last remaining node is the root of the Huffman tree.

48

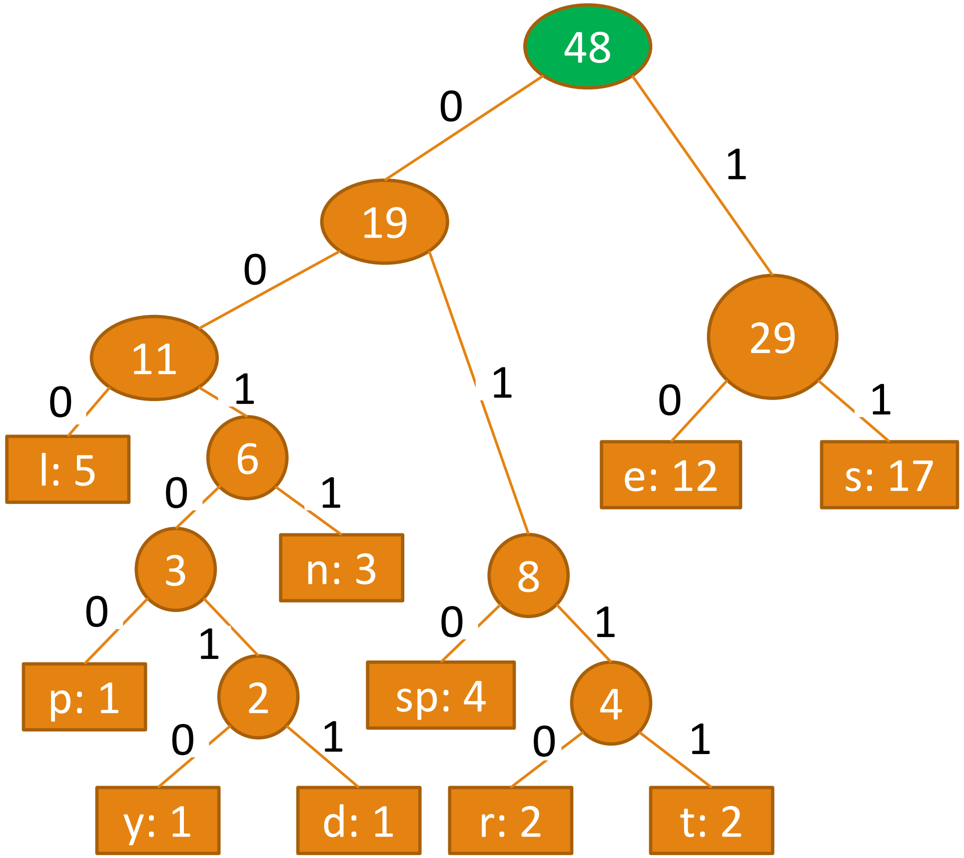
Huffman Coding - Example



Generate binary codes by traversing the Huffman tree.

Character	Codes
s	11
l	000
e	10
p	00100
n	0011
sp	010
t	0111
r	0110
d	001011
y	001010

Huffman Coding - Example



Generate binary codes by traversing the Huffman tree.

Char	Codes	Freq	Required Bits
s	11	17	2*17 = 34
l	000	5	3*5 = 15
e	10	12	2*12 = 24
p	00100	1	5*1 = 5
n	0011	3	4*3 = 12
sp	010	4	3*4 = 12
t	0111	2	4*2 = 8
r	0110	2	4*2 = 8
d	001011	1	6*1 = 6
y	001010	1	6*1 = 6

Required Total Bits = **130** bits

Average Bits per Character

$$= \frac{130}{48}\,bits/char$$

$$= \mathbf{2.708}\ bits/char$$

Home Task:

Analysis the Time Complexity of Huffman Coding.

THANK YOU