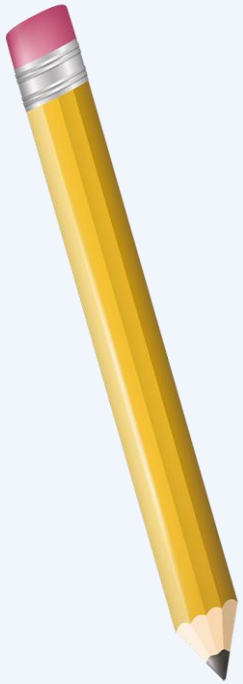
اللهم صل وسلم وبارك على سيدنا محمد وعلى
آله وصحبة وسلم تسليماً كثيراًطيباً مباركاً فيه

File Organization



Dr \ Mohammed Ahmed Mahfouz

Doctor of Information Systems
Thebes Higher Institute for Management and Information Technology

Organizing Files for Performance

Lecture No. 8

Contents

Data Compression

File Compression

- Reasons for file compression:
  - Less storage
  - Transmitting faster, decreasing access time
  - Processing faster sequentially

Why Data Compression?

- Make optimal use of limited storage space
- Save time and help to optimize resources
  - If compression and decompression are done in I/O processor, less time is required to move data to or from storage subsystem, freeing I/O bus for other work
  - In sending data over communication line: less time to transmit and less storage to host

Data Compression- Entropy

- **Entropy** is the measure of information content in a message.
- Messages with higher entropy carry more information than messages with lower entropy.
- How to determine the entropy:
  - Find the probability p(x) of symbol x in the message
  - The entropy H(x) of the symbol x is:

$$H(x) = - p(x) \bullet \log_2 p(x)$$

- The average entropy over the entire message is the sum of the entropy of all n symbols in the message

Cont. Data Compression- Entropy

1. Meaning of the Formula:

$$H(x) = -p(x) \cdot Log_2 p(x)$$

- This formula is used toType equation here. calculate the **amount of information (Information Content)** or **entropy** associated with a symbol x that appears with a certain probability.
- ✔ **What do the terms mean?**
- **p(x)**
  The probability of the symbol x.
  Example: If symbol **A** appears 4 times out of 10,

$$p(A) = \frac{4}{10} = 0.4$$

- $Log_2 p(x)$
  The logarithm of the probability using **base 2**
  (we use base 2 because information is measured in **bits**).

Cont. Data Compression- Entropy

- **H(x)**
  The amount of information carried by symbol x.
  A **rare symbol** → conveys **more information**.
  A **frequent symbol** → conveys **less information**.
- **The negative sign (−):**
  Probabilities are between 0 and 1 → their log base 2 is **negative**.
  The negative sign makes entropy positive.

2. Intuition Behind the Formula

- If a symbol is **rare** → p(x) is small → Log_2(p(x)) is strongly negative →H(x) is large ⇒ The symbol carries **a lot of information** because its appearance is surprising.
- If a symbol is **very common** → p(x) is large → Log_2(p(x)) is close to zero →H(x) is small ⇒ The symbol carries **little information** because its appearance is expected.

Cont. Data Compression- Entropy

3. Full Example

- Consider the following message:

  A A A B B C

- Total number of symbols = 6
- **Compute probabilities**

  A appears 3 times → p(A)=3/6=0.5

  B appears 2 times → p(B)=2/6=0.33

  C appears 1 time → p(C)=1/6=0.167

4. Compute H(x) for Each Symbol

1) Entropy of A

- $H(A)=-0.5 \cdot Log_2(0.5)$

  We know: $Log_2(0.5)=-1$

  Therefore: $H(A)=-0.5 \cdot (-1)=0.5$ bit

Cont. Data Compression- Entropy

2) Entropy of B

$H(A) = -0.33 \cdot Log_2(0.33)$

We know: $Log_2(0.33) \approx -1.585$

Therefore: $H(A) = -0.33 \cdot (-1.585) \approx 0.523$ bit

3) Entropy of C

$H(A) = -0.167 \cdot Log_2(0.167)$

We know: $Log_2(0.167) = \approx -2.585$

Therefore: $H(A) = -0.167 \cdot (-2.585) = \approx 0.432$ bit

5. Total Entropy of the Message

H=H(A)+H(B)+H(C)HH=0.5+0.523+0.432=1.455 bits/symbol

✔ **Interpretation:** On average, the minimum number of bits needed to encode each symbol is: ≈1.46 bits per symbol

This is the theoretical lower bound that **optimal compression algorithms** such as **Huffman coding** try to achieve.

Cont. Data Compression- Entropy

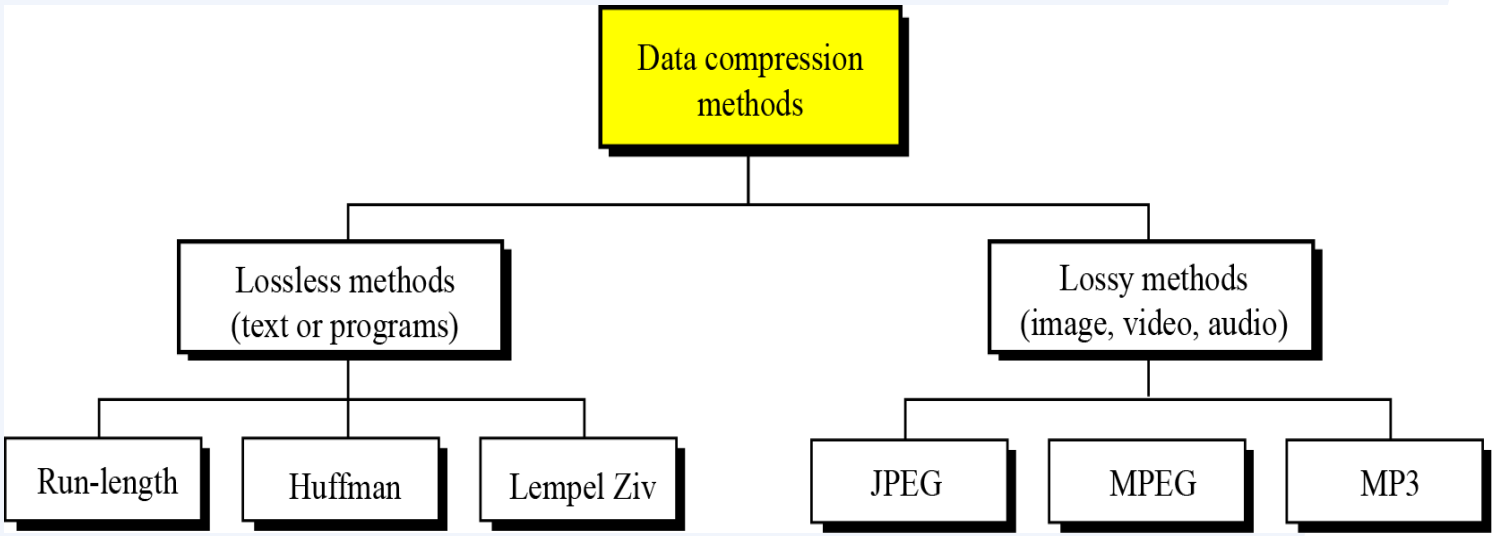
6. Simple Summary

- A symbol that appears often → contains **little information**.
- A symbol that is rare → contains **a lot of information**.
- Entropy is the **average amount of information per symbol**.
- Entropy helps us understand **how much we can compress data**.
- The final entropy value (1.46 bits/symbol here) tells us the best possible compression limit.

Data Compression Methods

- Data compression is about storing and sending a smaller number of bits.

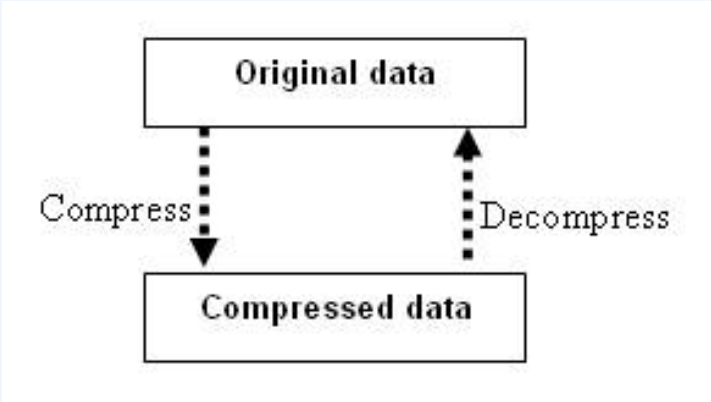
- There're two major categories for methods to compress data: lossless and lossy methods



Lossless Compression Methods

Lossless Compression Methods

- In lossless methods, original data and the data after compression and decompression are exactly the same.
- Redundant data is removed in compression and added during decompression.
- Lossless methods are used when we can't afford to lose any data: legal and medical documents, computer programs.

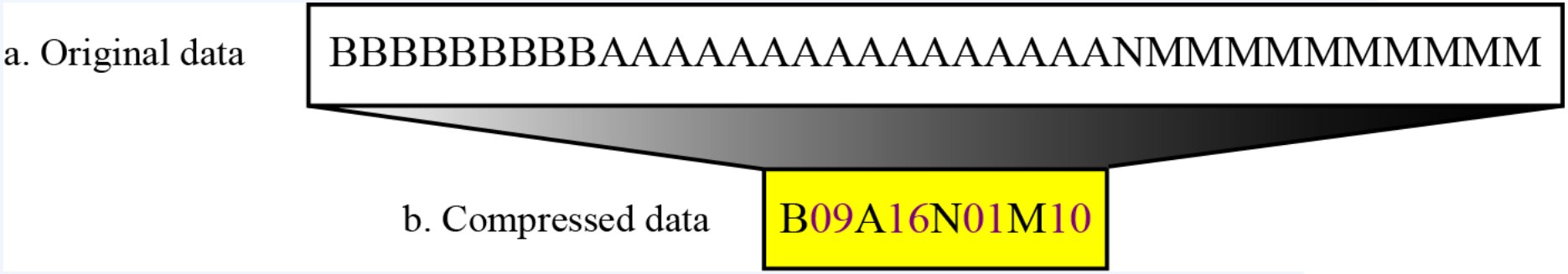


Run-length Encoding

Run-length Encoding

- Simplest method of compression.

- Replace consecutive repeating occurrences of a symbol by one occurrence of the symbol itself, then followed by the number of occurrences.



Run-length Encoding

$$\begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 255 & 0 & 255 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 255 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 255 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 255 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 255 \end{bmatrix}$$

Run-length Encoding

- **Run-length encoding algorithm:**
  - Read through pixels, copying pixel values to file in sequence, except the same pixel value occurs more than once in succession
  - When the same value occurs more than once in succession, substitute the following three bytes:
    - special run-length code indicator(e.g. 0xFF)
    - pixel value repeated
    - the number of times that value is repeated

- **Example:**

  22 23 24 24 24 24 24 24 24 25 26 26 26 26 26 26 25 24

  RL-coded stream: 22 23 ff 24 07 25 ff 26 06 25 24

Run-length Encoding

- **Run-length encoding algorithm:**
  - It is an example of redundancy reduction
  - **Drawbacks:**
    - not guarantee any particular amount of space savings
    - under some circumstances, compressed image is larger than original image
    - Why? Can you prevent this?

Huffman Coding

Huffman Coding

- Assign fewer bits to symbols that occur more frequently and more bits to symbols appear less often.

- There's no unique Huffman code and every Huffman code has the same average code length.

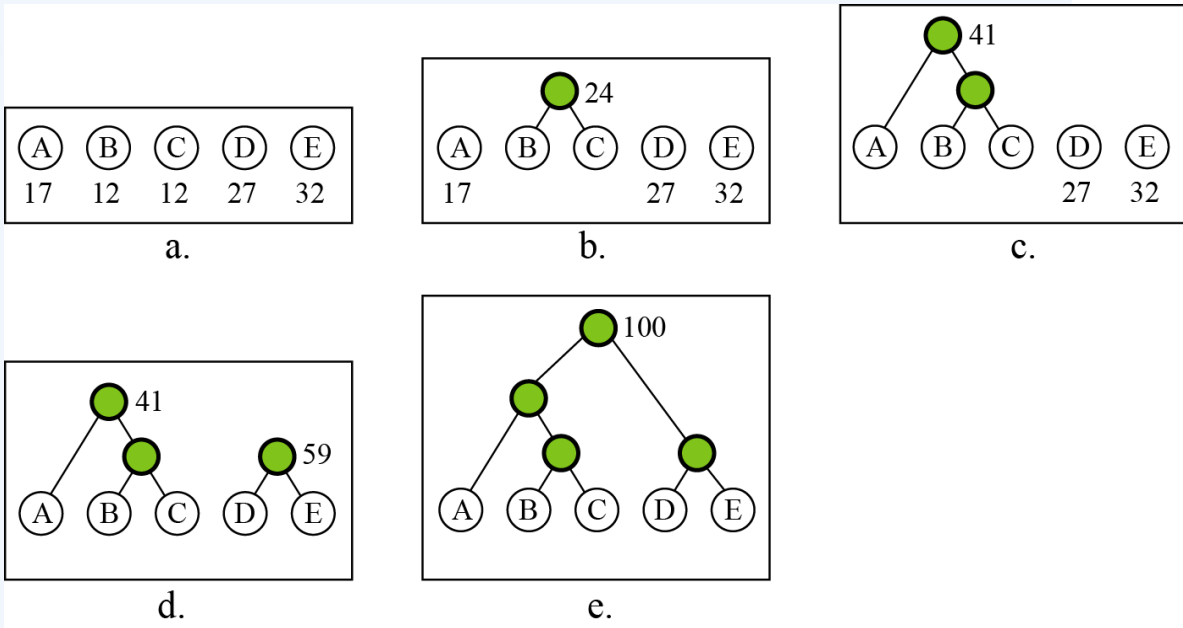
Huffman Coding

- **Algorithm**:
  1. Make a leaf node for each code symbol
     - Add the generation probability of each symbol to the leaf node
  2. Take the two leaf nodes with the smallest probability and connect them into a new node
     - Add 1 or 0 to each of the two branches
     - The probability of the new node is the sum of the probabilities of the two connecting nodes
  3. If there is only one node left, the code construction is completed. If not, go back to (2)

Huffman Coding

- Example:

2

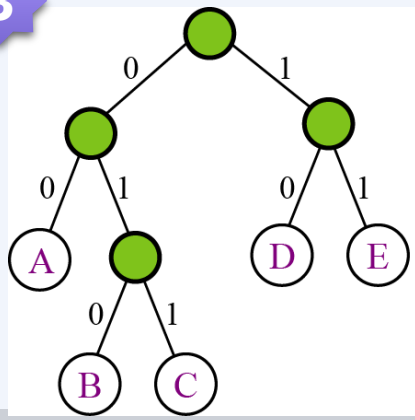


1

Table 15.1 Frequency of characters

Character	A	B	C	D	E
Frequency	17	12	12	27	32

3

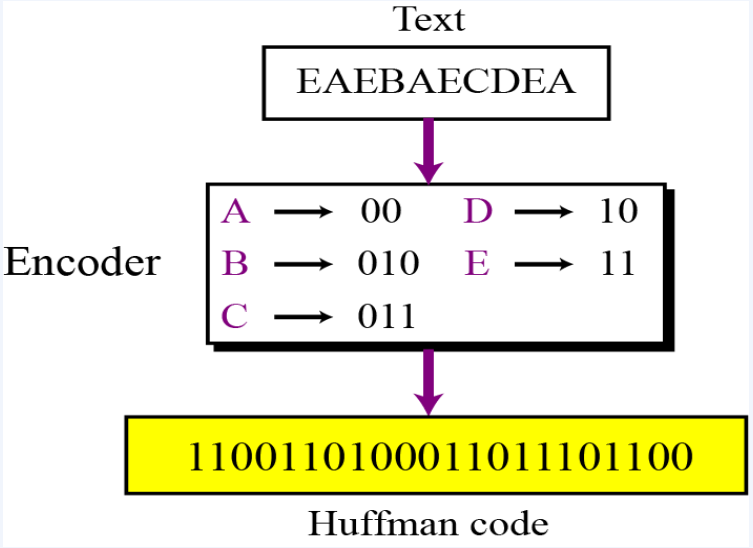


A: 00 D: 10
B: 010 E: 11
C: 011

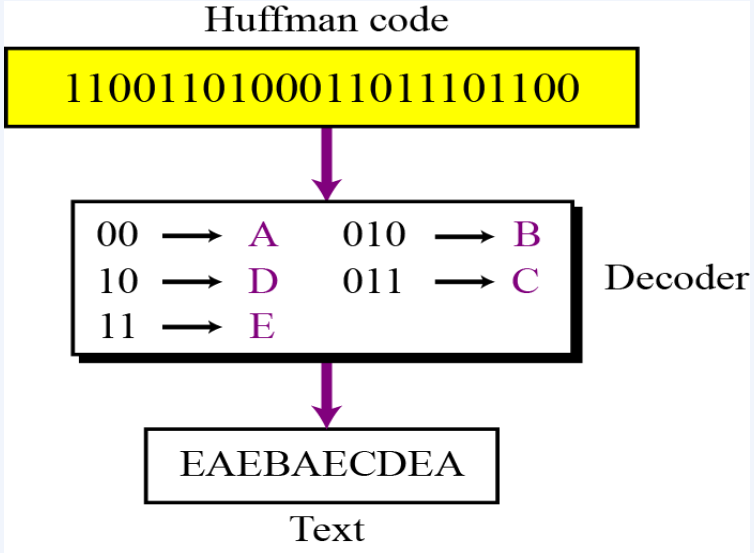
Code

Huffman Coding

- Encoding



- Decoding



Huffman Coding

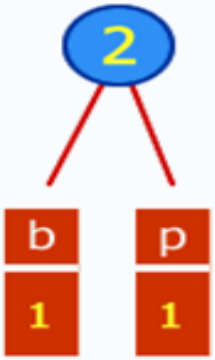
- Example:

File :

b	p	`	m	j	o	d	a	i	r	u	l	s	e	
1	1	2	2	3	3	3	4	4	5	5	6	6	8	12

File :

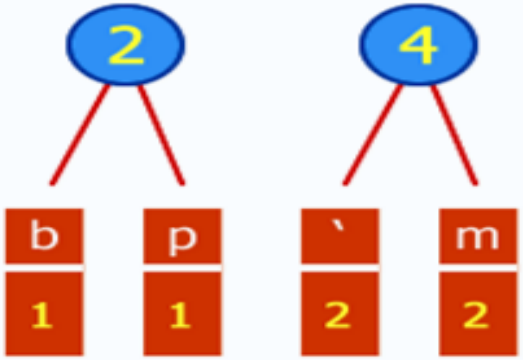
`	m	j	o	d	a	i	r	u	l	s	e	
2	2	3	3	3	4	4	5	5	6	6	8	12



Huffman Coding

File :

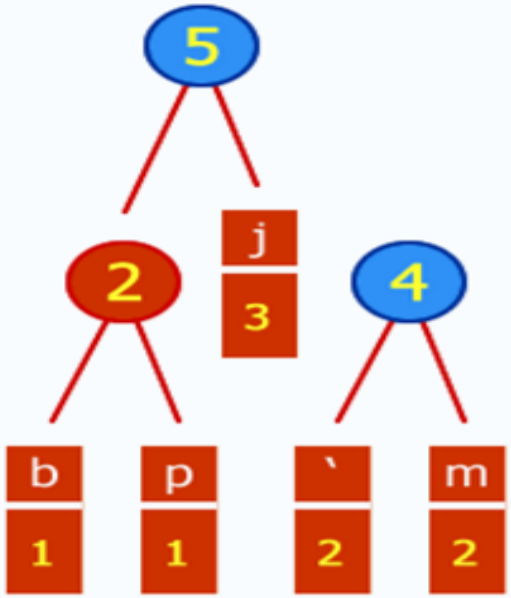
j	o	d	a	i	r	u	l	s	e	
3	3	3	4	4	5	5	6	6	8	12



Huffman Coding

File :

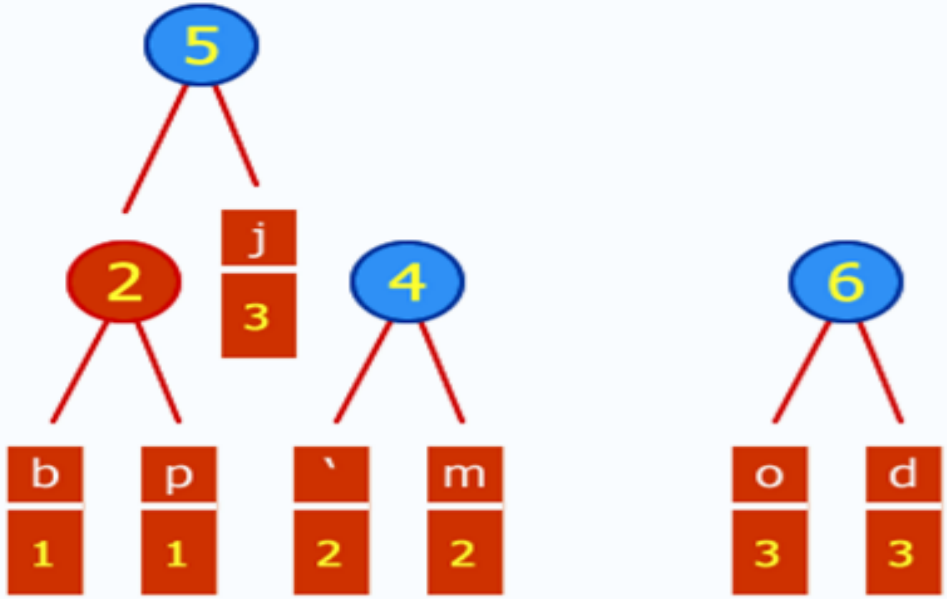
o	d	a	i	r	u	l	s	e	
3	3	4	4	5	5	6	6	8	12



Huffman Coding

File :

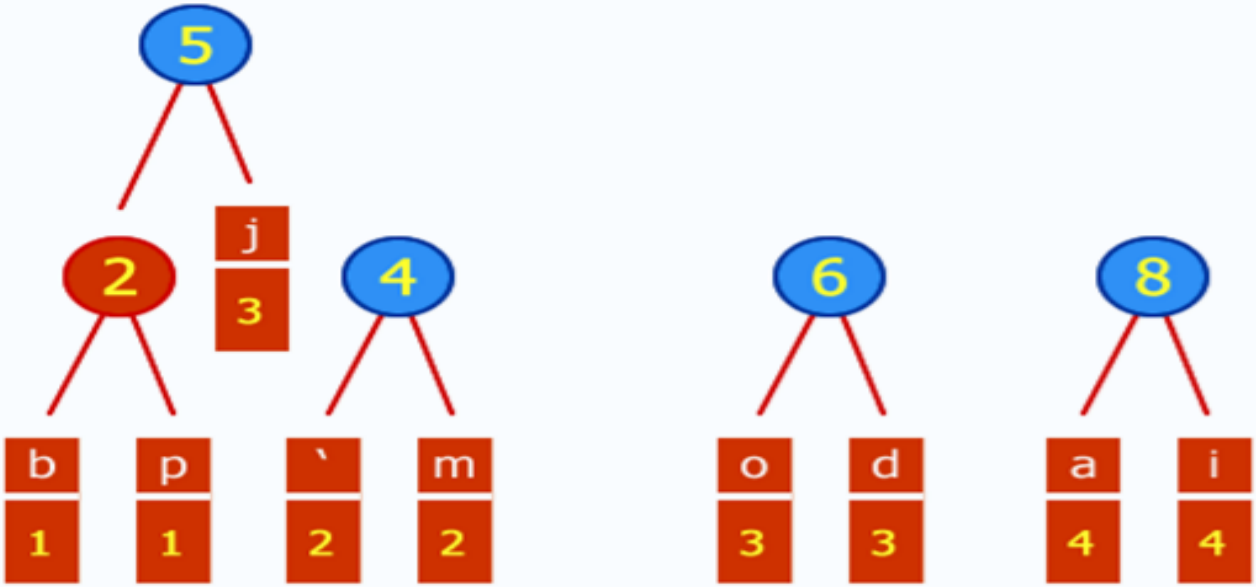
a	i	r	u	l	s	e	
4	4	5	5	6	6	8	12



Huffman Coding

File :

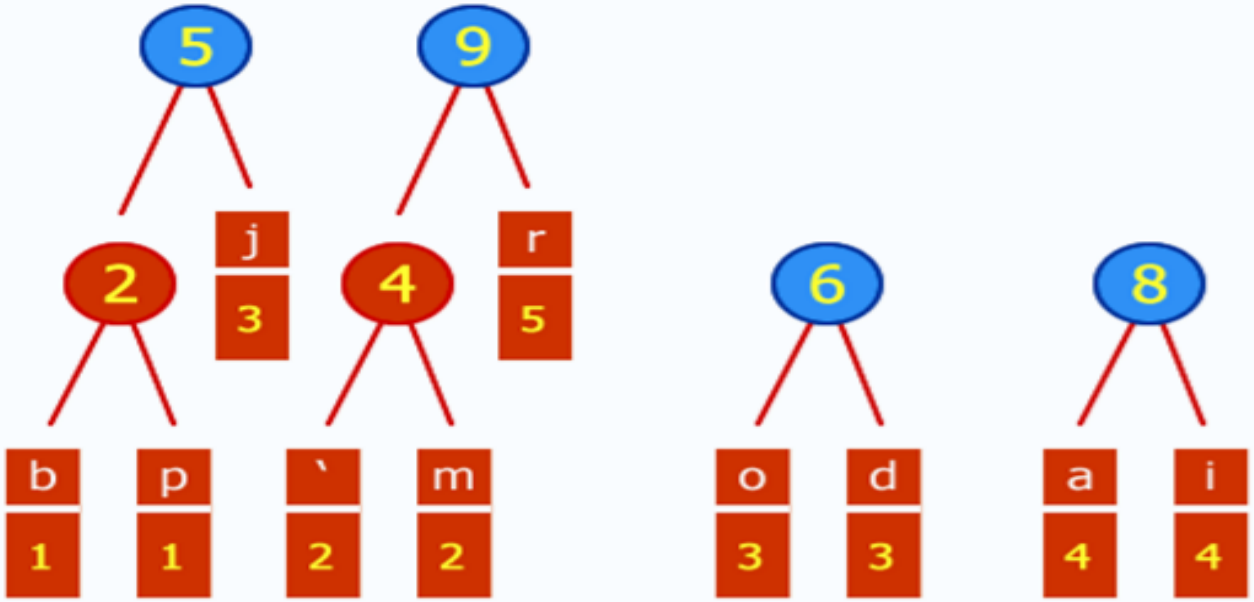
r	u	l	s	e	
5	5	6	6	8	12



Huffman Coding

File :

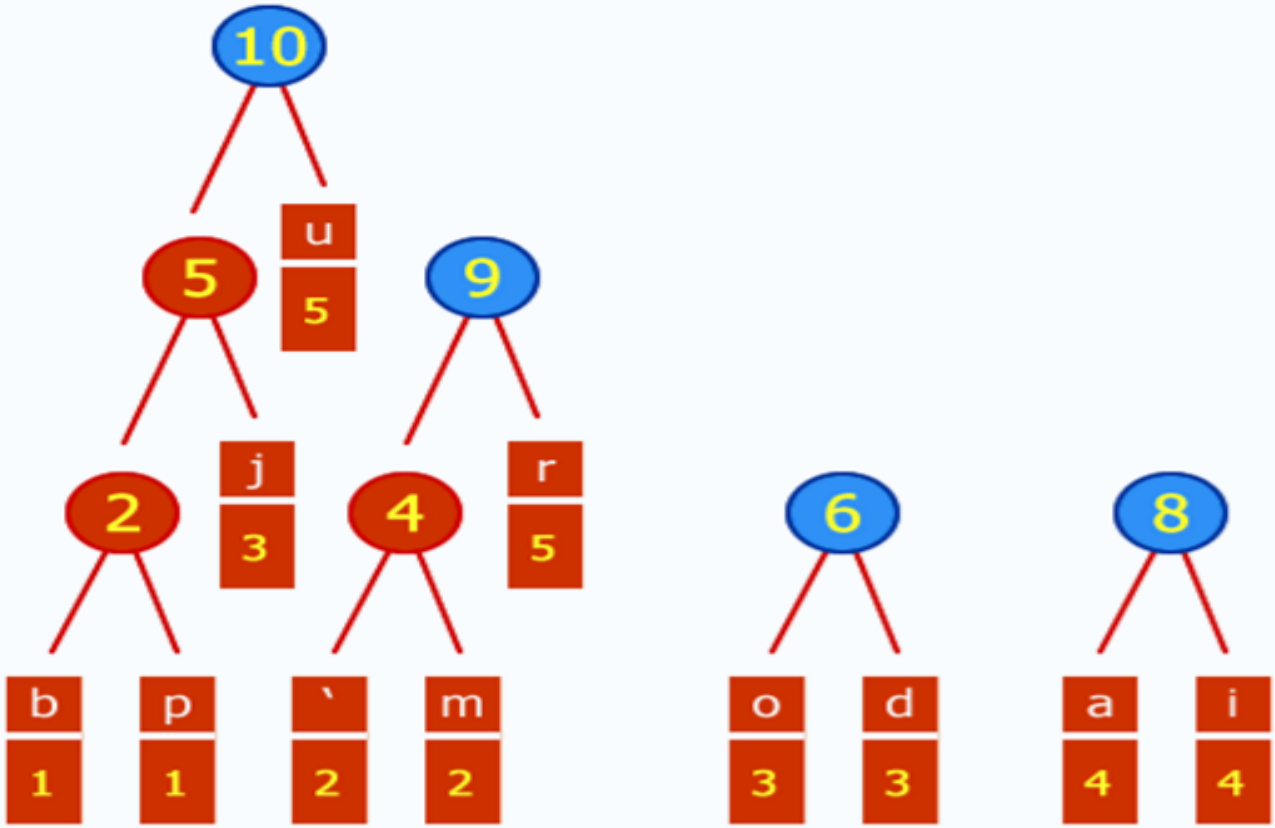
u	l	s	e	
5	6	6	8	12



Huffman Coding

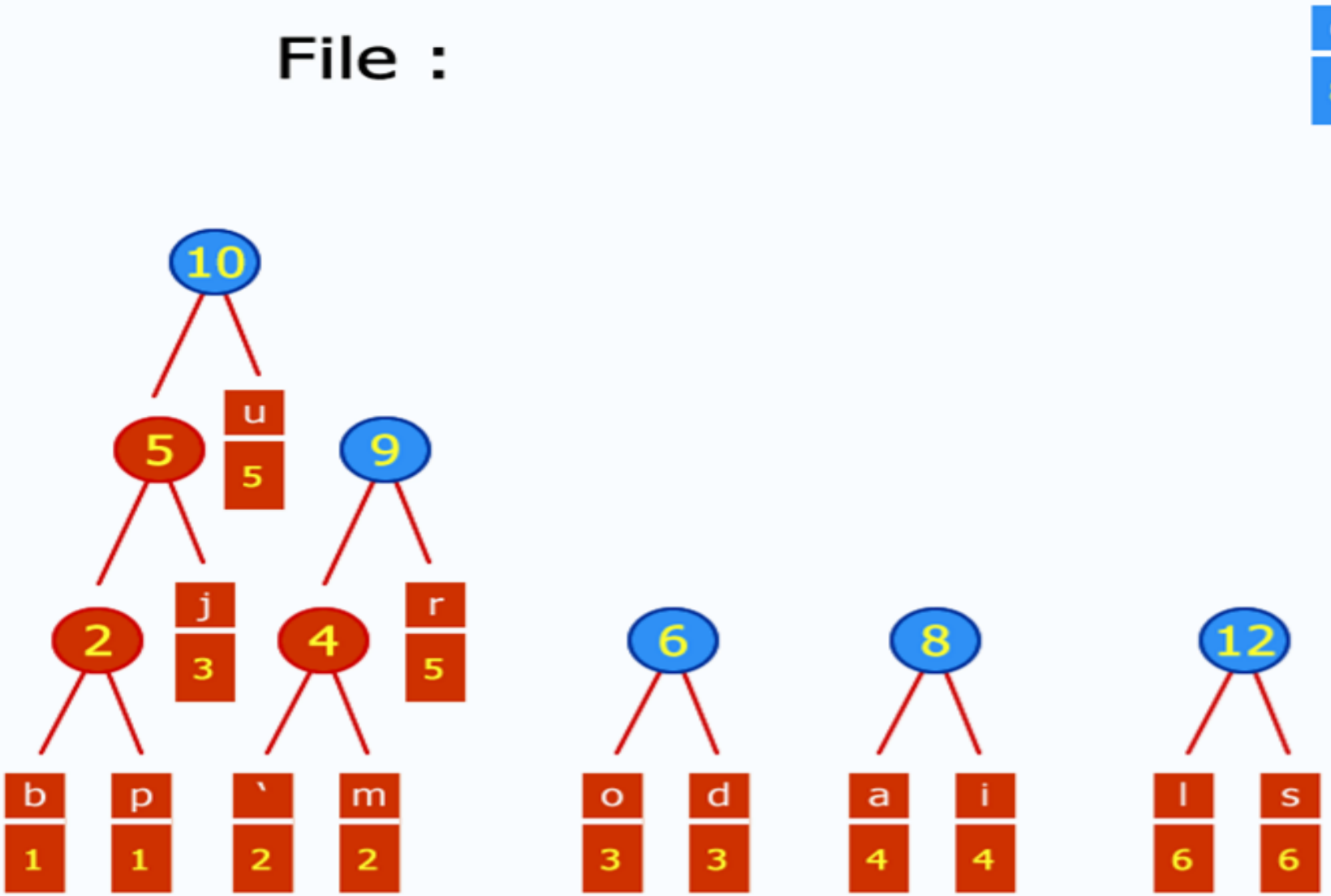
File :

l	s	e	
6	6	8	12



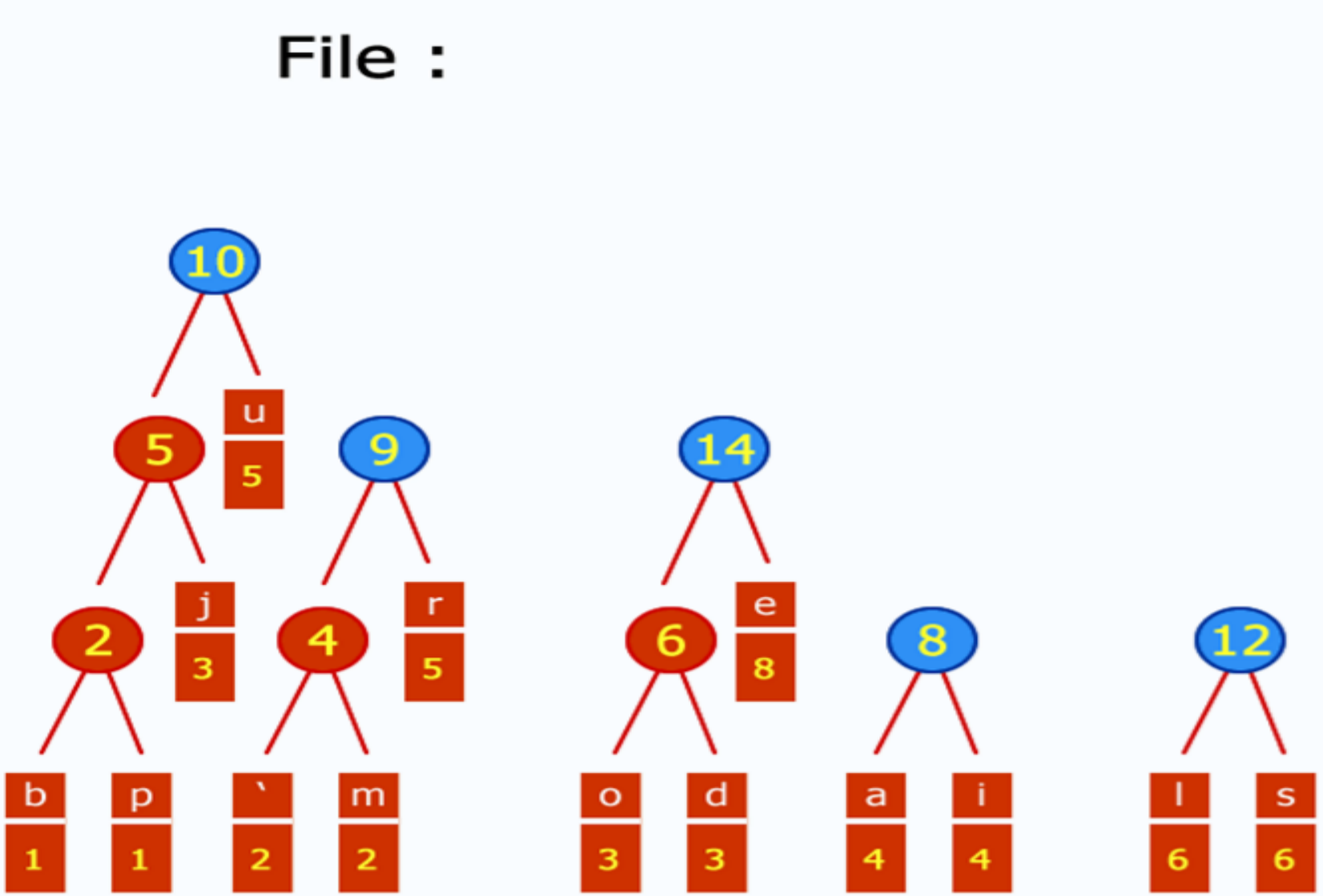
Huffman Coding

File :

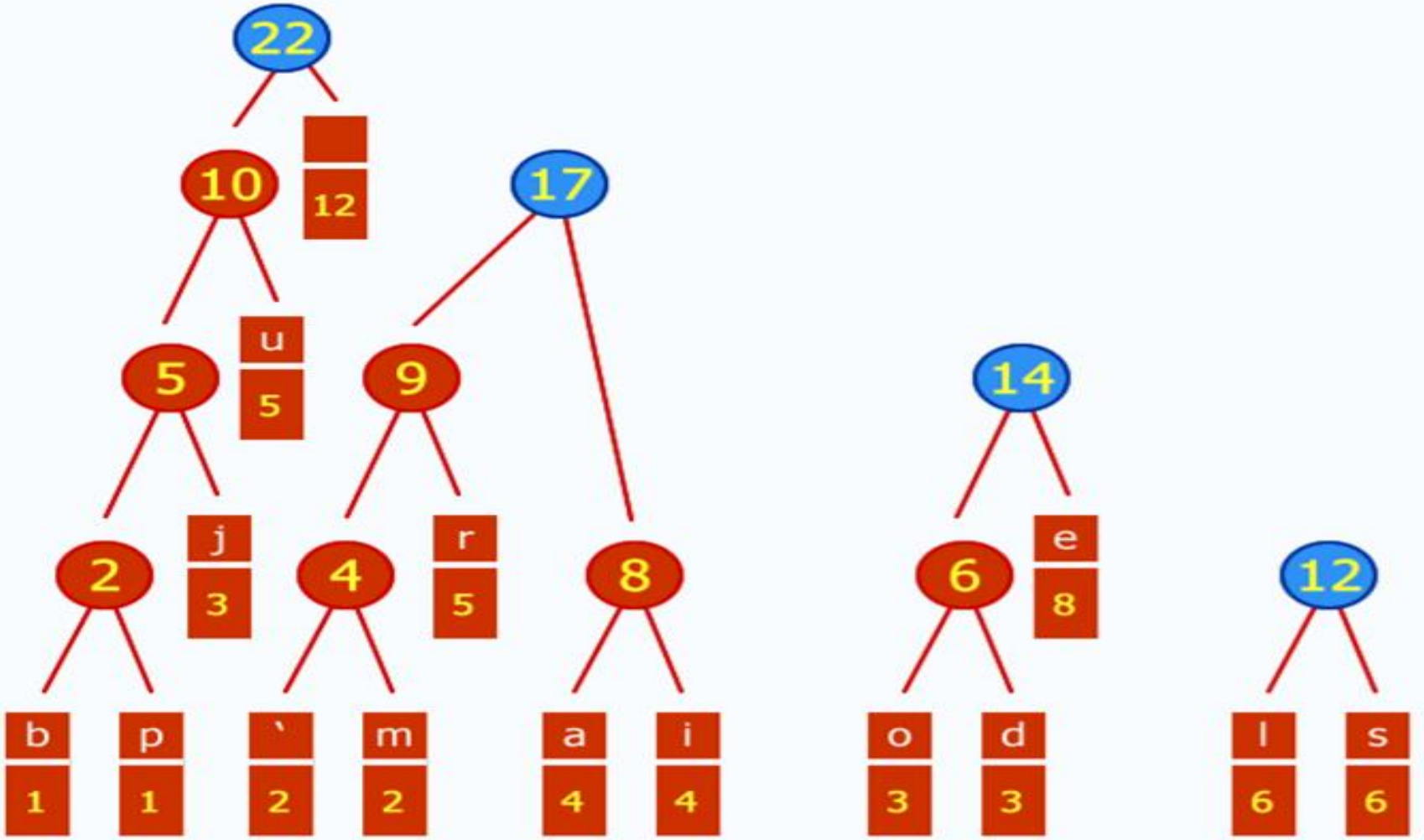


Huffman Coding

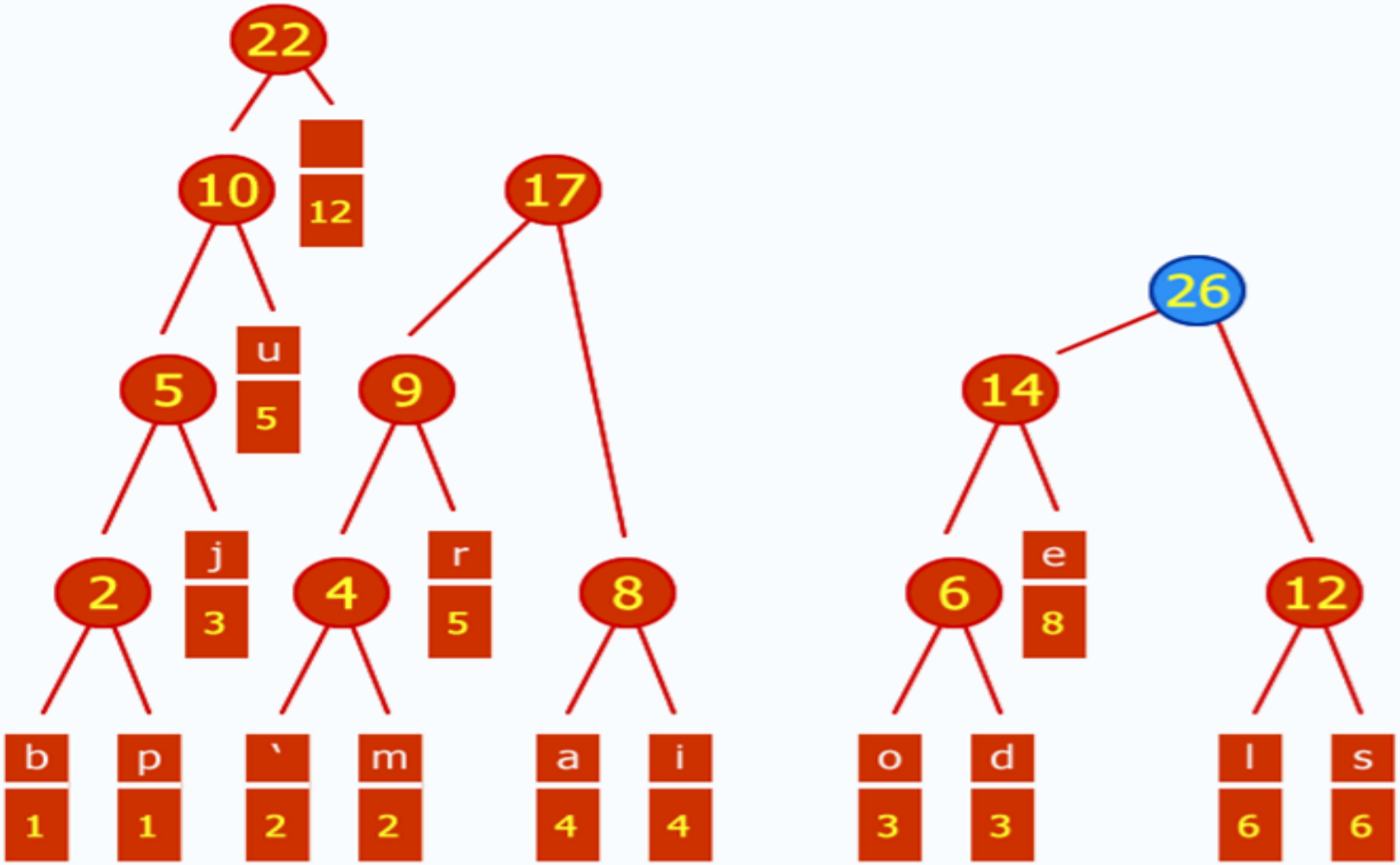
File :



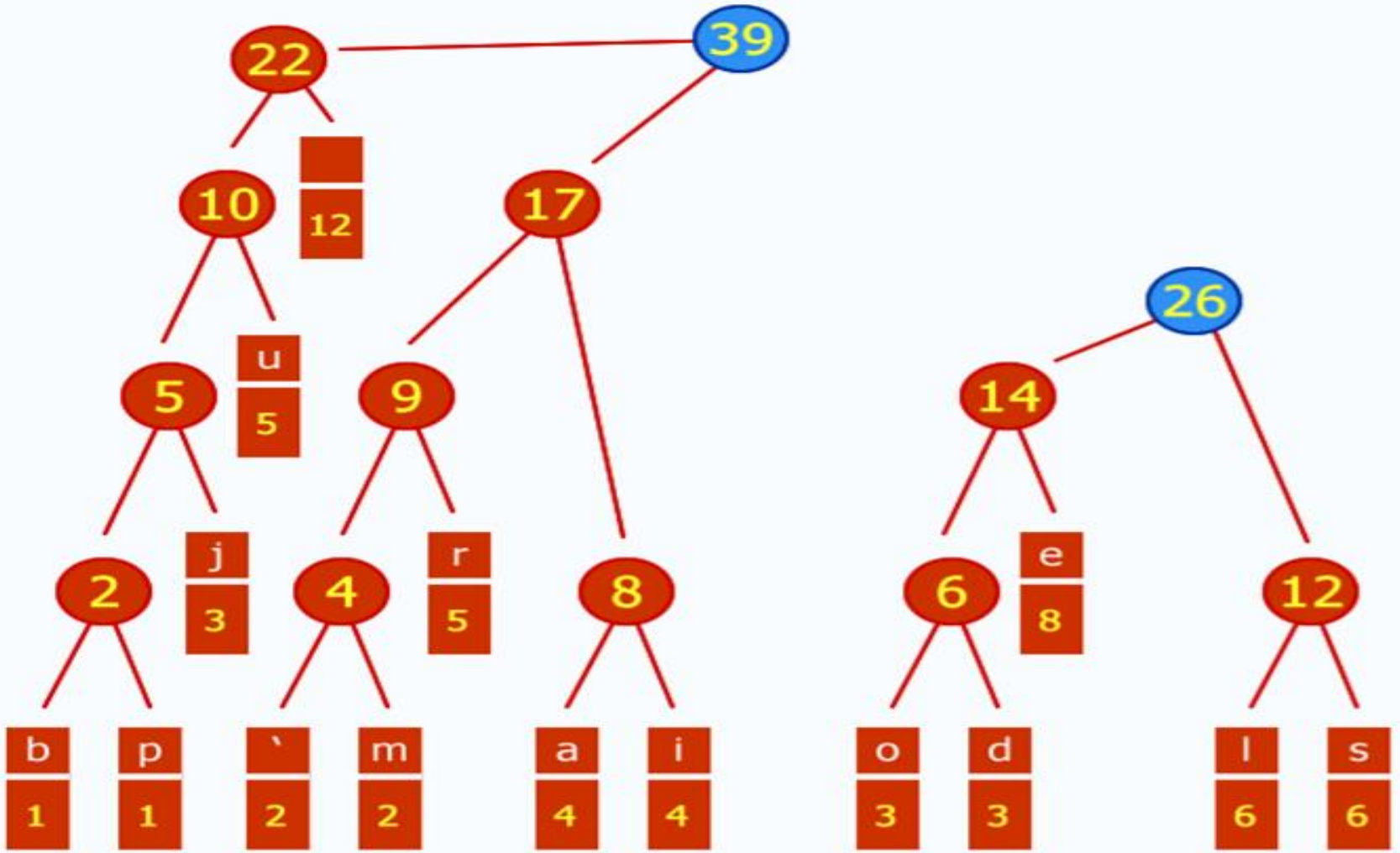
Huffman Coding



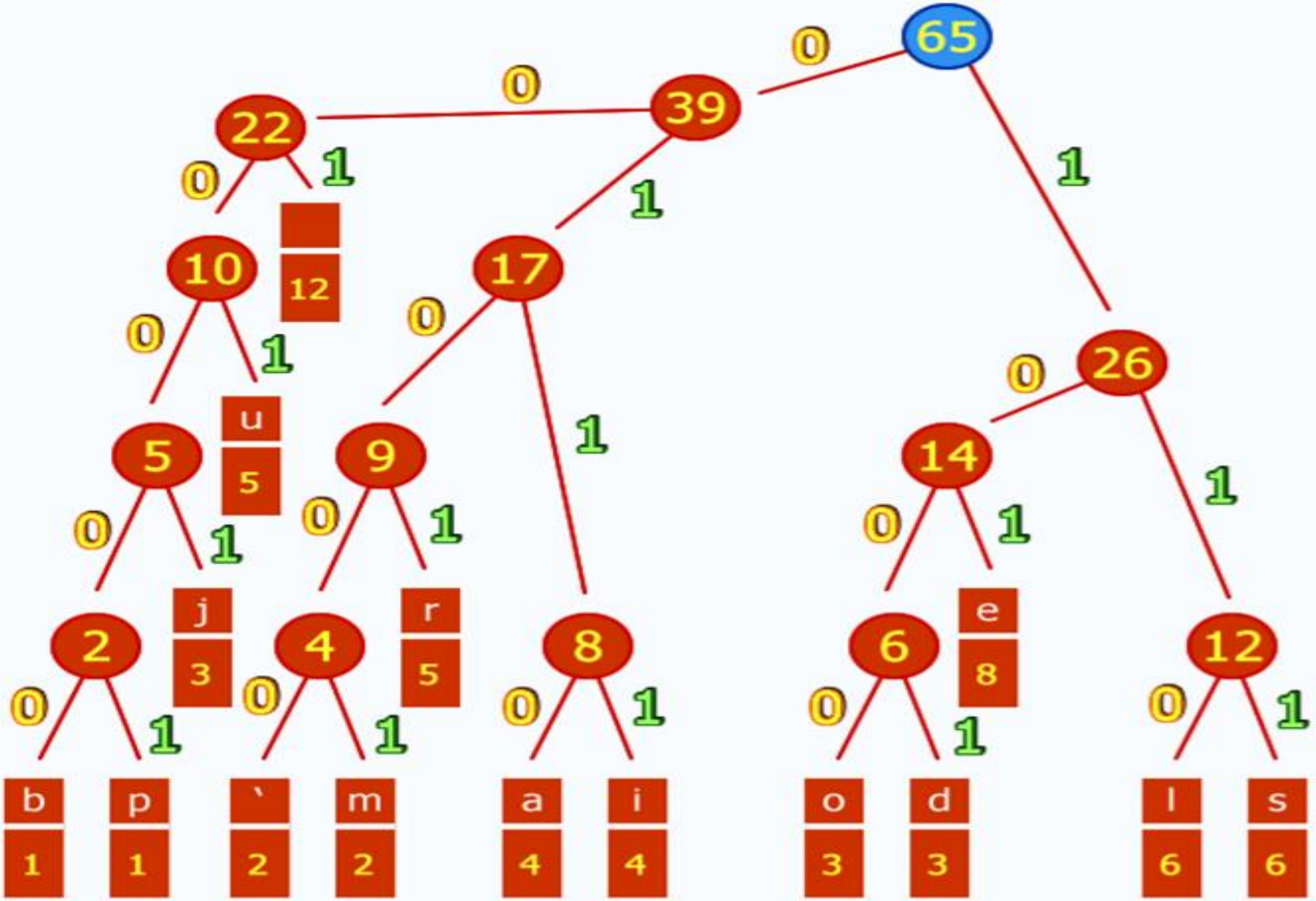
Huffman Coding



Huffman Coding



Huffman Coding



Lempel-Ziv Codes

Lempel-Ziv Codes

- There are several variations of Lempel-Ziv Codes.
- We will look at LZ78
- Commands zip and unzip and Unix compress and uncompress use Lempel-Ziv codes

Lempel-Ziv Codes

- Let us look at an example for an alphabet having only two letters:

  a a a b a b b b a a a b a a a a a a a b a a b b

- Rule
  - Separate this stream of characters into pieces of text so that each piece is the shortest string of characters that we have not seen yet.

a | a a | b | a b | b b | a a a | b a| a a a a | a a b | a a b b

Lempel-Ziv Codes

a a a b a b b b a a a b a a a a a a a b a a b b

a | a a | b | a b | b b | a a a | b a| a a a a | a a b | a a b b

1. We see “a”
2. “a” has been seen, we now see “aa”
3. We see “b”
4. “a” has been seen, we now see “ab”
5. “b” has been seen, we now see “bb”
6. “aa” has been seen, we now see “aaa”
7. “b” has been seen, we now see “ba”
8. “aaa” has been seen, we now see “aaaa”
9. “aa” has been seen, we now see “aab”
10. “aab” has been seen, we now see “aabb”

Lempel-Ziv Codes

- **Index:**
  - We have index values from 1 to n
  - For the previous example

1 2 3 4 5 6 7 8 9 10

a | a a | b | a b | b b | a a a | b a| a a a a | a a b | a a b b

Lempel-Ziv Codes

- **Encoding:**
  - Since each piece is the concatenation of a piece already seen with a new character, the message can be encoded by a previous index plus a new character.

1 2 3 4 5 6 7 8 9 10

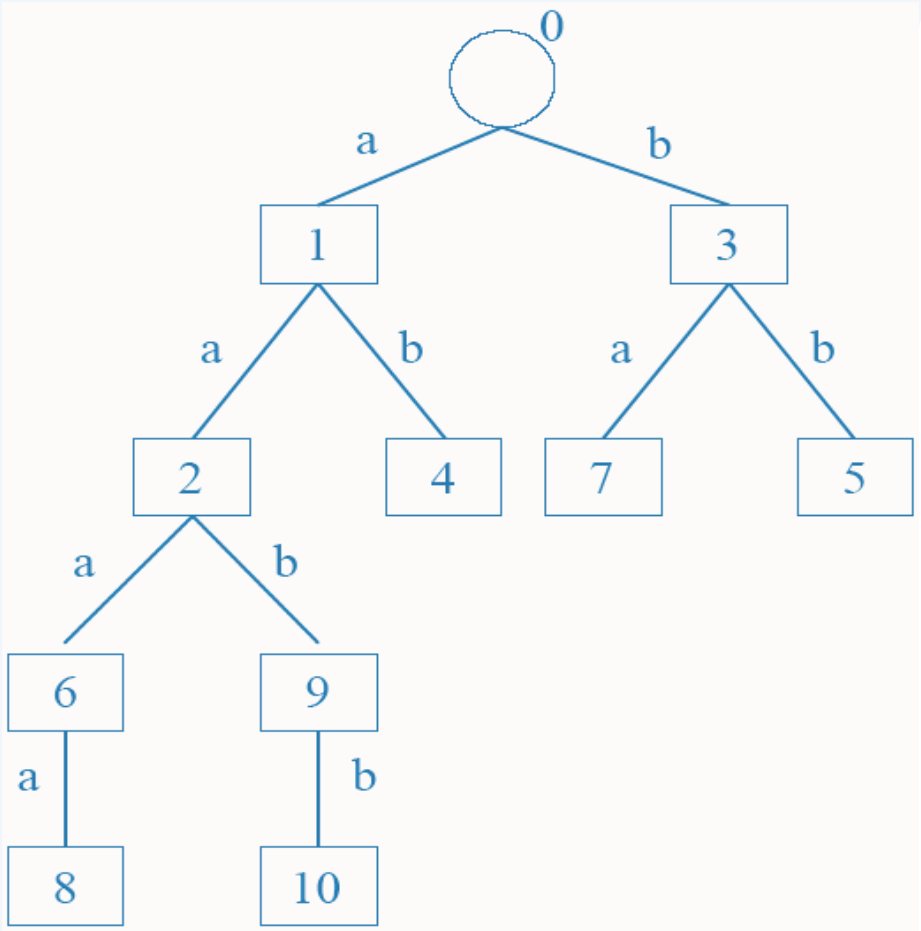
a | a a | b | a b | b b | a a a | b a| a a a a | a a b | a a b b

1 2 3 4 5 6 7 8 9 10

0 a | 1 a | 0 b | 1 b | 3 b | 2 a | 3 a | 6 a | 2 b | 9 b

Lempel-Ziv Codes

- **Encoding tree:**
  - A tree can be built when encoding



1	2	3	4	5	6	7	8	9	10
a	a a	b	a b	b b	a a a	b a	a a a a	a a b	a a b b

Lempel-Ziv Codes

- **Exercise No. 1:**

  - Encode the file containing the following characters, drawing the corresponding digital tree.

“ a a a b b c b c d d d e a b ”

Lempel-Ziv Codes

- **Exercise No. 1:**

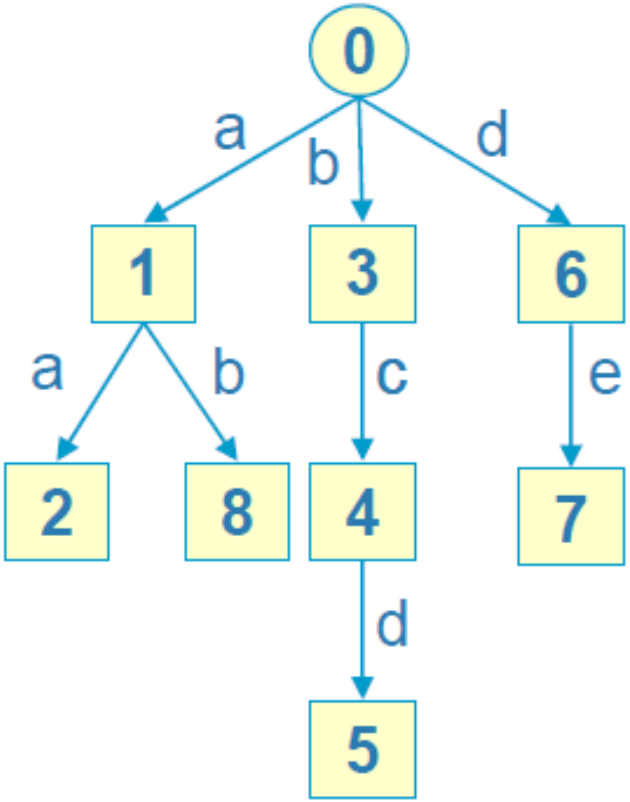
1 2 3 4 5 6 7 8

a | a a | b | b c | b c d | d | d e | a b

0 a | 1 a | 0 b | 3 c | 4 d | 0 d | 6 e | 1 b

Lempel-Ziv Codes

- **Exercise No. 1:**



1 2 3 4 5 6 7 8
a | a a | b | b c | b c d | d | d e | a b

0 a | 1 a | 0 b | 2 c | 4 d | 0 d | 6 e | 1 b

Lempel Ziv Encoding

- It is dictionary-based encoding

- **Basic idea:**
  - Create a dictionary (a table) of strings used during communication.
  - If both sender and receiver have a copy of the dictionary, then previously-encountered strings can be substituted by their index in the dictionary.

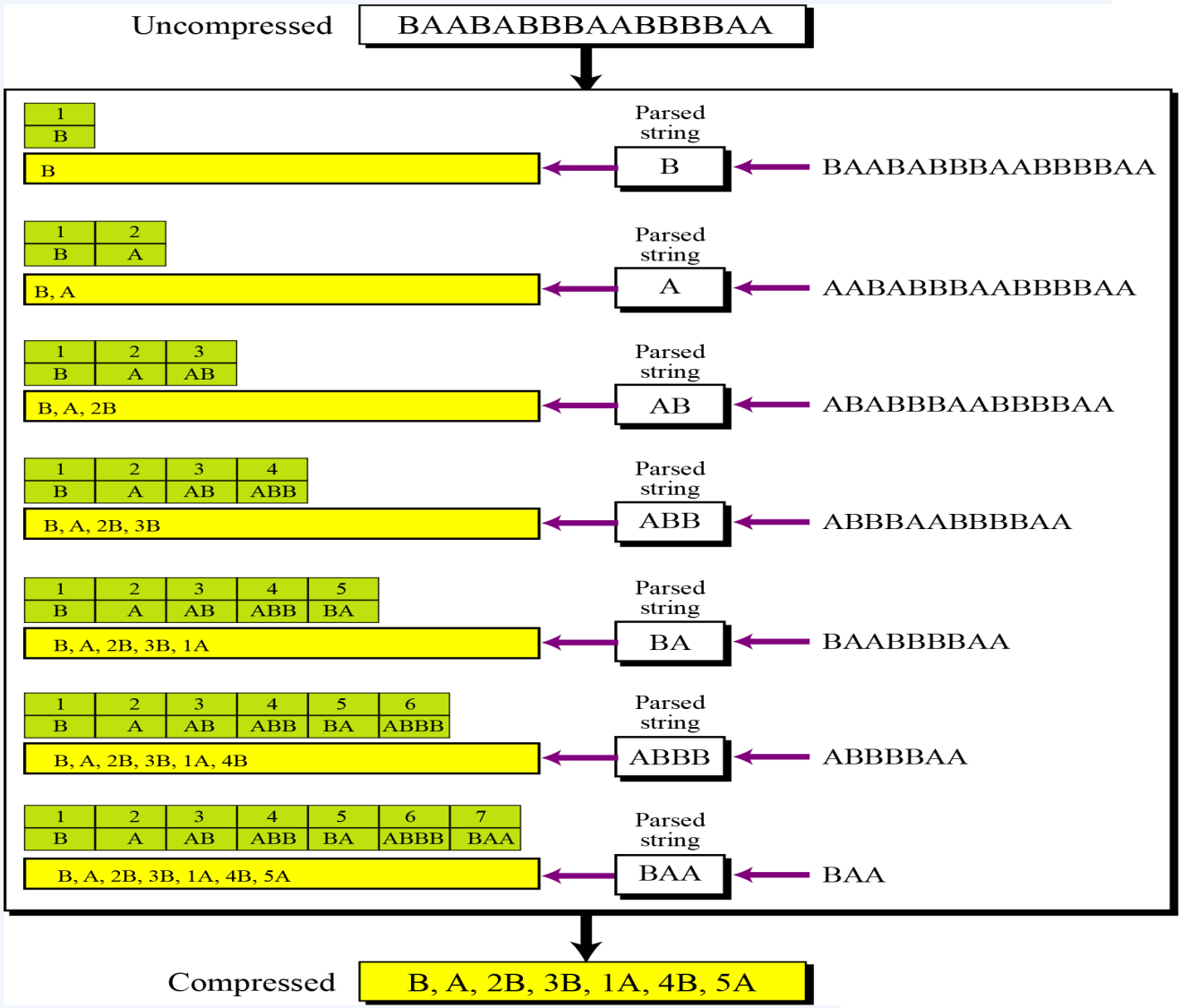
Lempel Ziv Compression

- **Have 2 phases:**
  1. Building an indexed dictionary
  2. Compressing a string of symbols

- **Algorithm:**
  1. Extract the smallest substring that cannot be found in the remaining uncompressed string.
  2. Store that substring in the dictionary as a new entry and assign it an index value
  3. Substring is replaced with the index found in the dictionary
  4. Insert the index and the last character of the substring into the compressed string

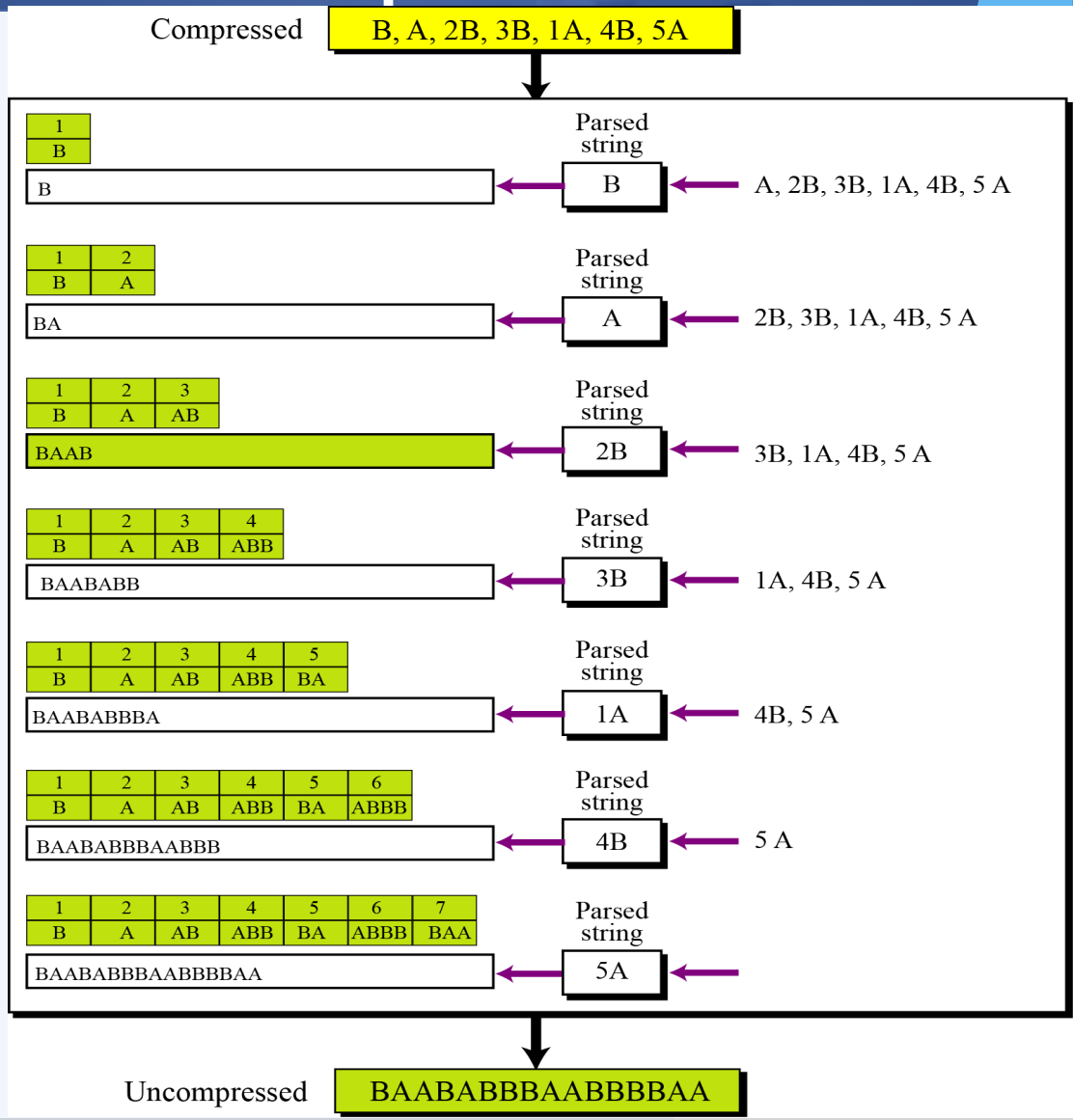
Lempel Ziv Compression

- Compression example:



Lempel Ziv Decompression

- It's just the inverse of compression process



Thank You !