Département Informatique Année Universitaire 2022-2023

Chapitre 3

Lossless Compression

Sommaire

- Compression
  - With loss
  - Without loss
- Shannon: Information Theory
- Shannon-Fano Coding Algorithm
- Huffman Coding Algorithm

Compression

- **Why Compression?**

All media, be it text, audio, graphics or video has “redundancy”. Compression attempts to eliminate this redundancy.

- **What is Redundancy?**

If one representation of a media content, M, takes X bytes and another takes Y bytes(Y< X), then X is a redundant representation relative to Y.

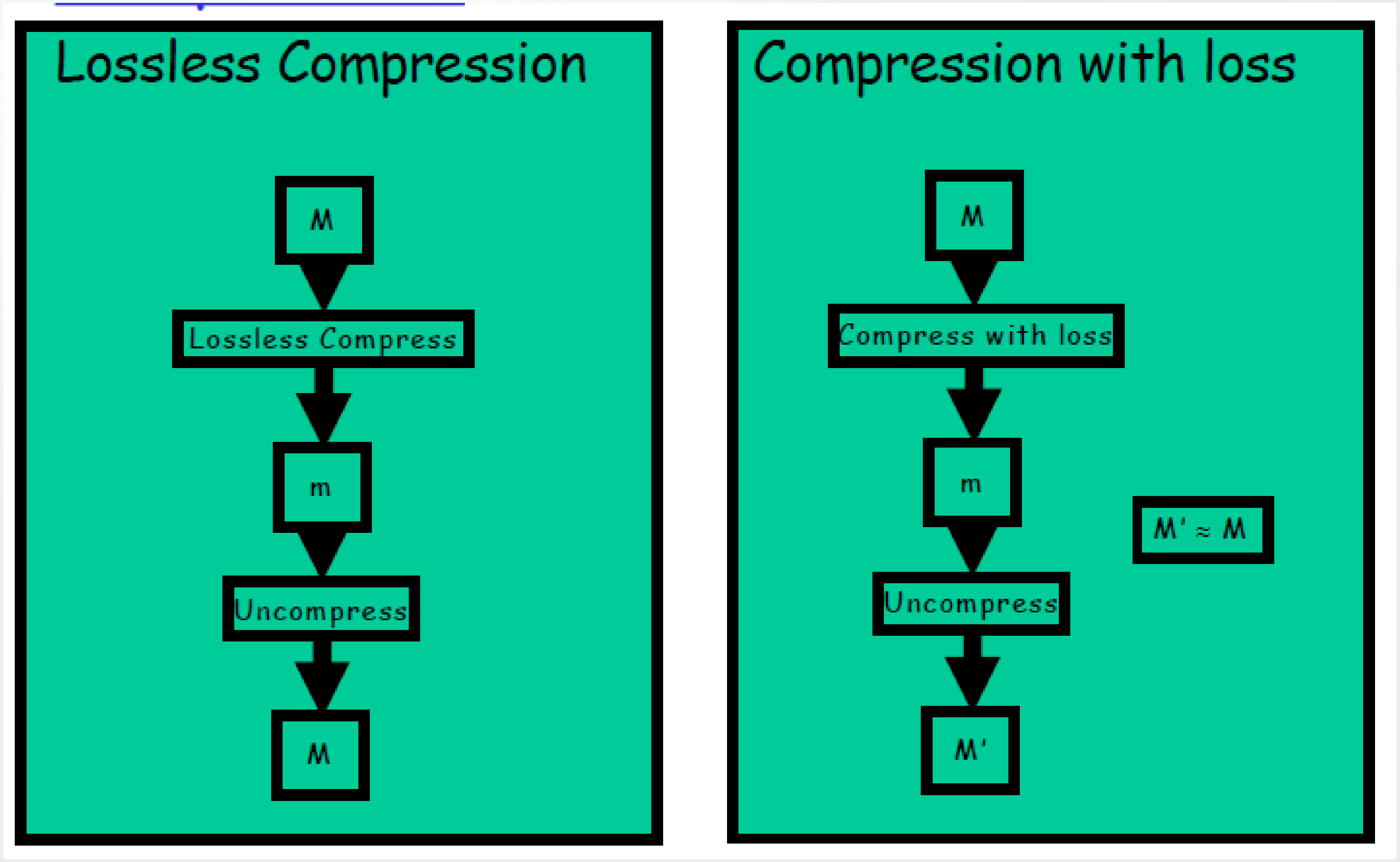
Other forms of Redundancy

If the representation of a media captures content that is not perceivable by humans, then removing such content will not affect the quality of the content.

- For example, capturing audio frequencies outside the human hearing range can be avoided without any harm to the audio’s quality.

Is there a representation with an optimal size Z that cannot be improved upon? This question is tackled by information theory.

Compression



Information Theory

According to Shannon, the **entropy of an information** source S is defined as:

$$H(S) = \Sigma_i (p_i \log_2 (1/p_i))$$

$\log_2 (1/p_i)$ indicates the amount of information contained in symbol S_i, i.e., the number of bits needed to code symbol S_i.

- For example, in an image with uniform distribution of gray level intensity, i.e. $p_i = 1/256$, with the number of bits needed to code each gray level being 8 bits. The entropy of the image is 8.

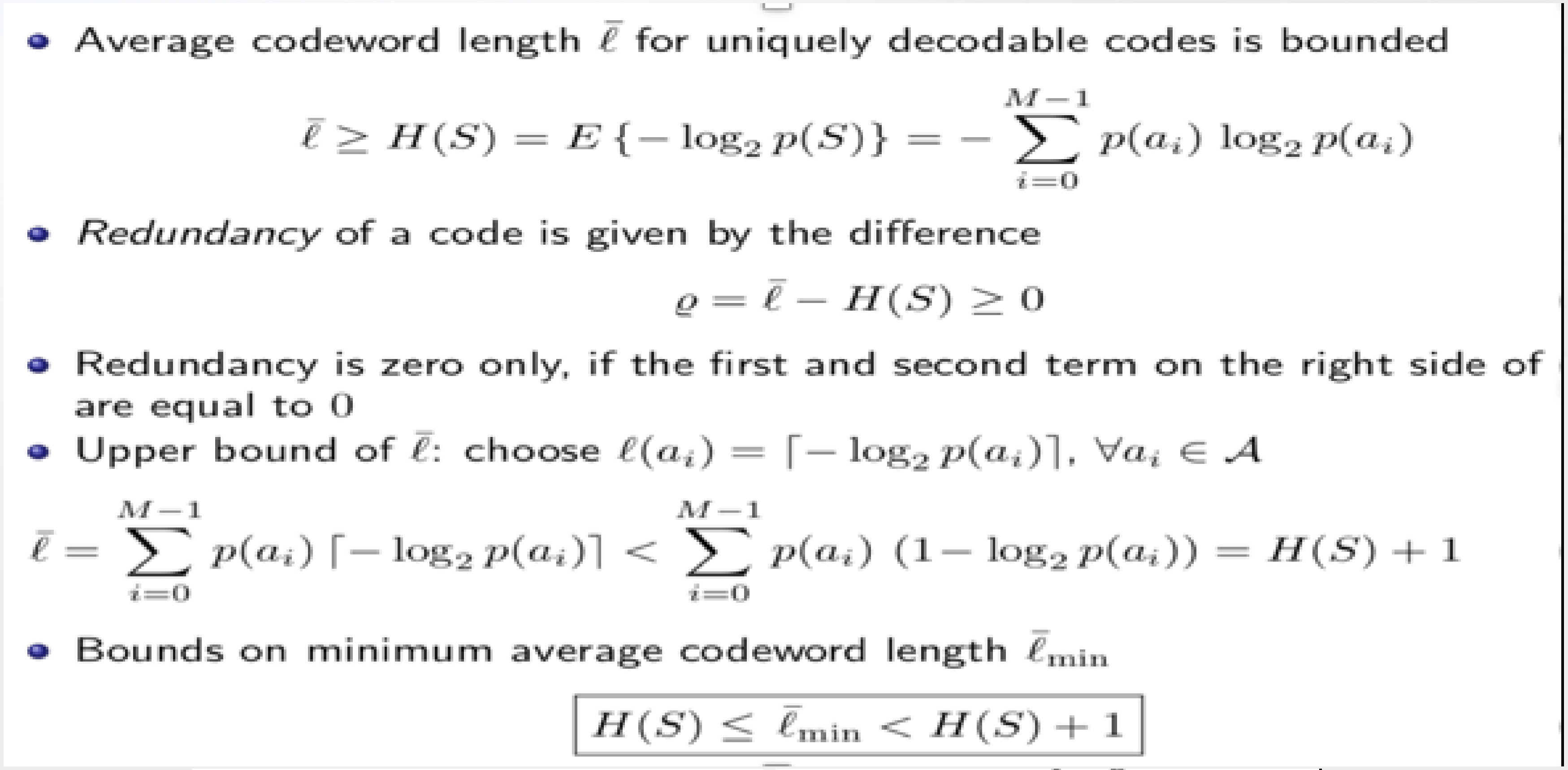
Q: What is the entropy of a source with M symbols where each symbol is equally likely?

- Entropy, $H(S) = \log_2 M$

Q: How about an image in which half of the pixels are white and half are black?

- Entropy, $H(S) = 1$

Entropy coding features



- Average codeword length $\bar{\ell}$ for uniquely decodable codes is bounded

$$\bar{\ell} \geq H(S) = E\{-\log_2 p(S)\} = -\sum_{i=0}^{M-1} p(a_i)\,\log_2 p(a_i)$$

- *Redundancy* of a code is given by the difference

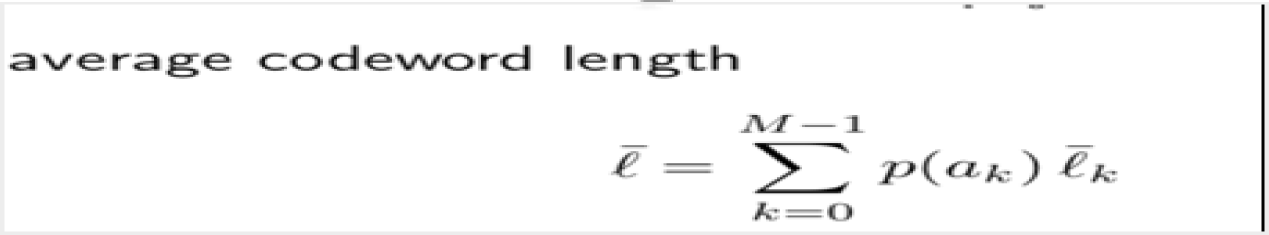
$$\varrho = \bar{\ell} - H(S) \geq 0$$

- Redundancy is zero only, if the first and second term on the right side of are equal to 0
- Upper bound of $\bar{\ell}$: choose $\ell(a_i) = \lceil -\log_2 p(a_i) \rceil$, $\forall a_i \in \mathcal{A}$

$$\bar{\ell} = \sum_{i=0}^{M-1} p(a_i)\,\lceil -\log_2 p(a_i) \rceil < \sum_{i=0}^{M-1} p(a_i)\,\left(1 - \log_2 p(a_i)\right) = H(S) + 1$$

- Bounds on minimum average codeword length $\bar{\ell}_{\min}$

$$\boxed{H(S) \leq \bar{\ell}_{\min} < H(S) + 1}$$



average codeword length

$$\bar{\ell} = \sum_{k=0}^{M-1} p(a_k)\,\bar{\ell}_k$$

Information Theory

Discussion:

Entropy is a measure of how much information is encoded in a message. Higher the entropy, higher the information content.

- We could also say entropy is a measure of uncertainty in a message. Information and Uncertainty are equivalent concepts.

The units (in coding theory) of entropy are bits per symbol.

- It is determined by the base of the logarithm:

  2: binary (bit);

  10: decimal (digit).

Entropy gives the actual number of bits of information contained in a message source.

- **Example: If the probability of the character 'e' appearing in**

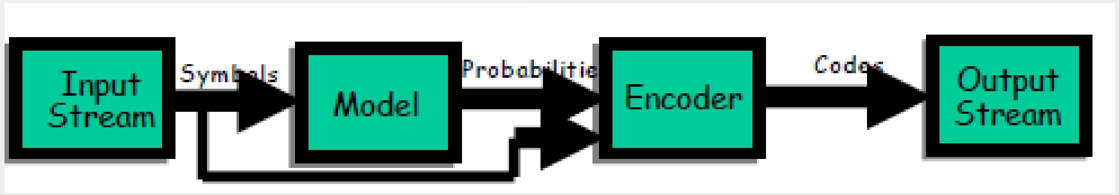
this slide is 1/16, then the information content of this character is 4 bits. So, the character string "eeeee" has a total content of 20 bits (contrast this to using an 8-bit ASCII coding that could result in 40 bits to represent "eeeee").

Data Compression = Modeling + Coding

Data Compression consists of taking a stream of symbols and transforming them into codes.

- The model is a collection of data and rules used to process input symbols and determine their probabilities.
- A coder uses a model (probabilities) to spit out codes when its given input symbols

Let's take Huffman coding to demonstrate the distinction:



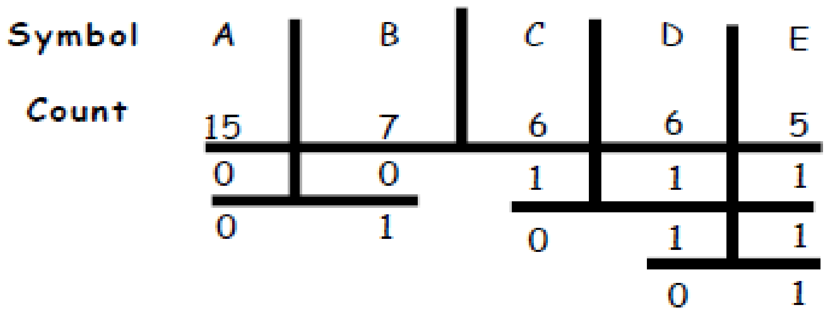
The output of the Huffman encoder is determined by the Model (probabilities). Higher the probability shorter the code.

- Model A could determine raw probabilities of each symbol occurring anywhere in the input stream.
    - *(pi = # of occurrences of Si / Total number of Symbols)*
- Model B could determine prob. based on the last 10 symbols in the i/p stream. (continuously re-computes the probabilities)

The Shannon-Fano Encoding Algorithm

1. Calculate the frequencies of the list of symbols (organize as a list).
2. Sort the list in (decreasing) order of frequencies.
3. Divide list into two halfs, with the total freq. *Counts of each half being as close as possible to the other.*
4. The upper half is assigned a code of 0 and lower a code of 1.
5. Recursively apply steps 3 and 4 to each of the halves, until each symbol has become a corresponding code leaf on a tree.

Example



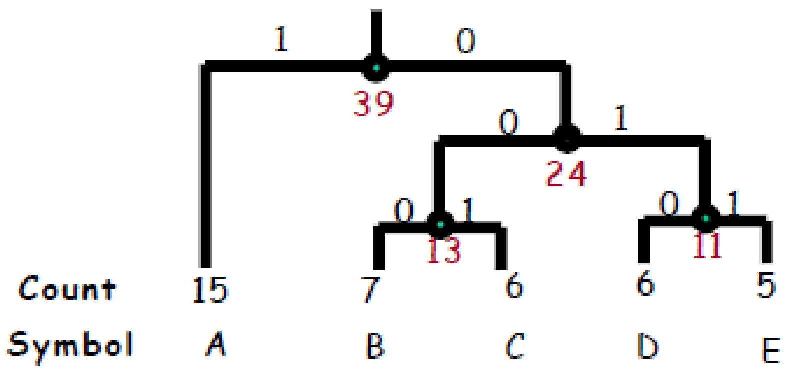
Symbol	Count		Info. -log2(pi)	Code	Subtotal # of Bits
A	15	x	1.38	00	30
B	7	x	2.48	01	14
C	6	x	2.70	10	12
D	6	x	2.70	110	18
E	5	x	2.96	111	15
	85.25				89

It takes a total of 89 bits to encode 85.25 bits of information (Pretty good huh!)

The Huffman Algorithm

1. Initialization: Put all nodes in an OPEN list *L*, keep it sorted at all times (e.g., ABCDE).
2. Repeat the following steps until the list *L* has only one node left:
    1. From *L* pick two nodes having the lowest frequencies, create a parent node of them.
    2. Assign the sum of the children's frequencies to the parent node and insert it into L.
    3. Assign code 0, 1 to the two branches of the tree, and delete the children from L.

Example



Symbol	Count	Info. -log2(pi)	Code	Subtotal # of Bits
A	15 x	1.38	1	15
B	7 x	2.48	000	21
C	6 x	2.70	001	18
D	6 x	2.70	010	18
E	5 x	2.96	011	15
	85.25			87

Huffman Alg.: Discussion

Decoding for the above two algorithms is trivial as long as the coding table (the statistics) is sent before the data. There is an overhead for sending this, negligible if the data file is big.
Unique Prefix Property: no code is a prefix to any other code (all symbols are at the leaf nodes)
--> *great for decoder, unambiguous;* ***unique Decipherability?***

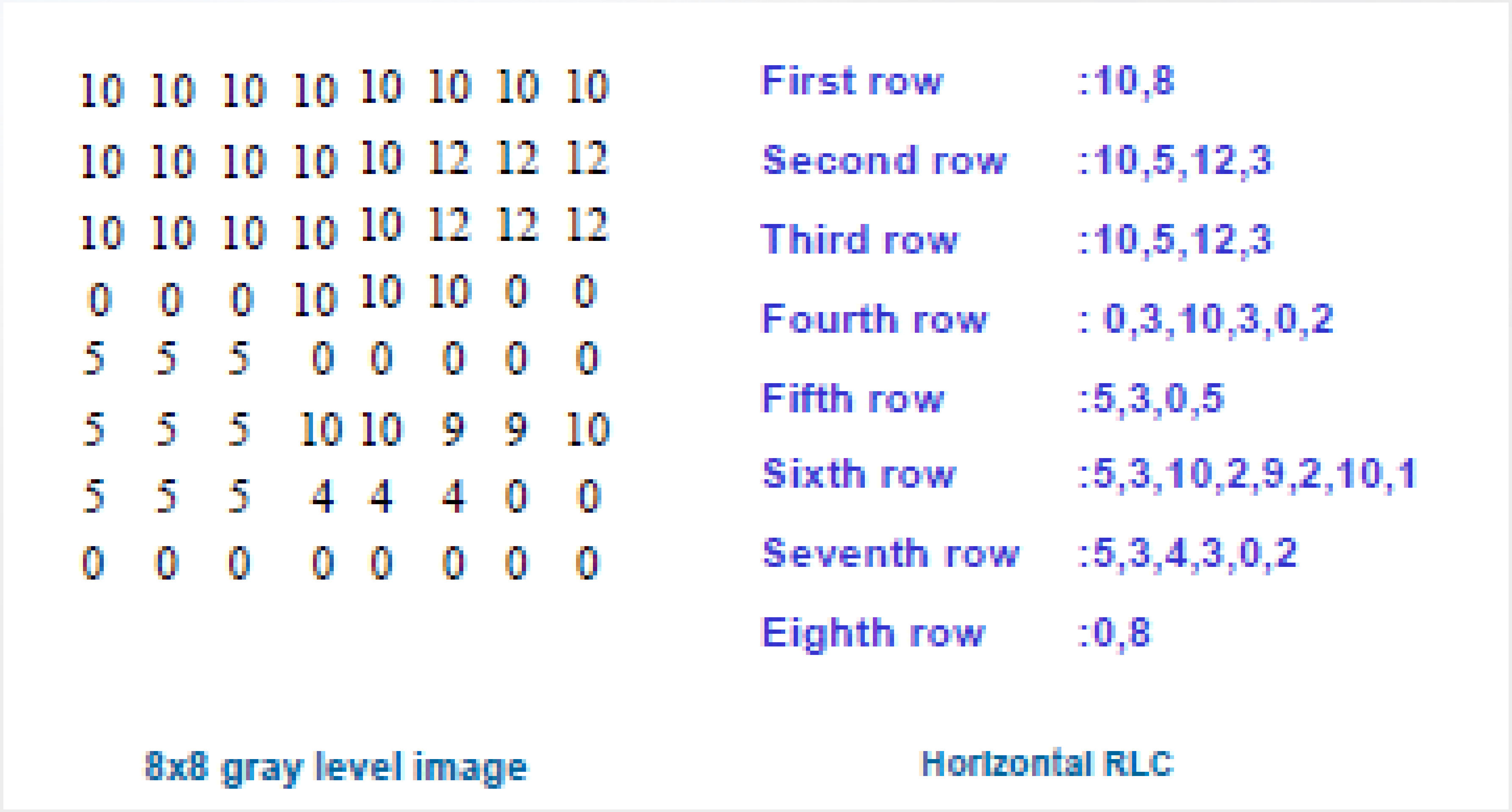
If prior statistics are available and accurate, then Huffman coding is very good.

- Number of bits (per symbol) needed for Huffman Coding is:
- 87 / 39 = 2.23
- Number of bits (per symbol)needed for Shannon-Fano
- Coding is:
- 89 / 39 = 2.28

Codage à longueur statique (RLC)

- **RLC (Run-Length Coding)**
  - La forme la plus simple de compression de données
- **Principe:**
  - SI un ensemble de symboles tend à former un groupe**, coder un symbole ainsi que la longueur du groupe,** au lieu de coder chaque symbole.
- **Exemple:** Une image à deux niveaux de gris, peut-être codée par RLC.
- RLC à deux dimensions est souvent utilisé pour coder une image à 2 niveaux de gris.

Codage à longueur statique (RLC)



10	10	10	10	10	10	10	10
10	10	10	10	10	12	12	12
10	10	10	10	10	12	12	12
0	0	0	10	10	10	0	0
5	5	5	0	0	0	0	0
5	5	5	10	10	9	9	10
5	5	5	4	4	4	0	0
0	0	0	0	0	0	0	0

8x8 gray level image

First row	:10,8
Second row	:10,5,12,3
Third row	:10,5,12,3
Fourth row	: 0,3,10,3,0,2
Fifth row	:5,3,0,5
Sixth row	:5,3,10,2,9,2,10,1
Seventh row	:5,3,4,3,0,2
Eighth row	:0,8

Horizontal RLC

Definitions

- **Alphabet:** is a collection of symbols.
- **Letters (symbols):** is an element of an alphabet.
- **Coding:** the assignment of binary sequences to elements of an alphabet.
- **Code:** A set of binary sequences.
- **Codewords:** Individual members of the set of binary sequences.

Examples of Binary Codes

- English alphabets:
  - 26 uppercase and 26 lowercase letters and punctuation
  - marks.
  - ASCII code for the letter “a” is 1000011
  - ASCII code for the letter “A” is 1000001
  - ASCII code for the letter “,” is 0011010
- **Note:**
  - All the letters (symbols) in this case use the same number of bits (7). These are called fixed length codes.
  - The average number of bits per symbol (letter) is called the rate of the code.

Code Rate

- Average length of the code is important in compression.
- Suppose our source alphabet consists of four letters a1, a2, a3, and a4 with probabilities

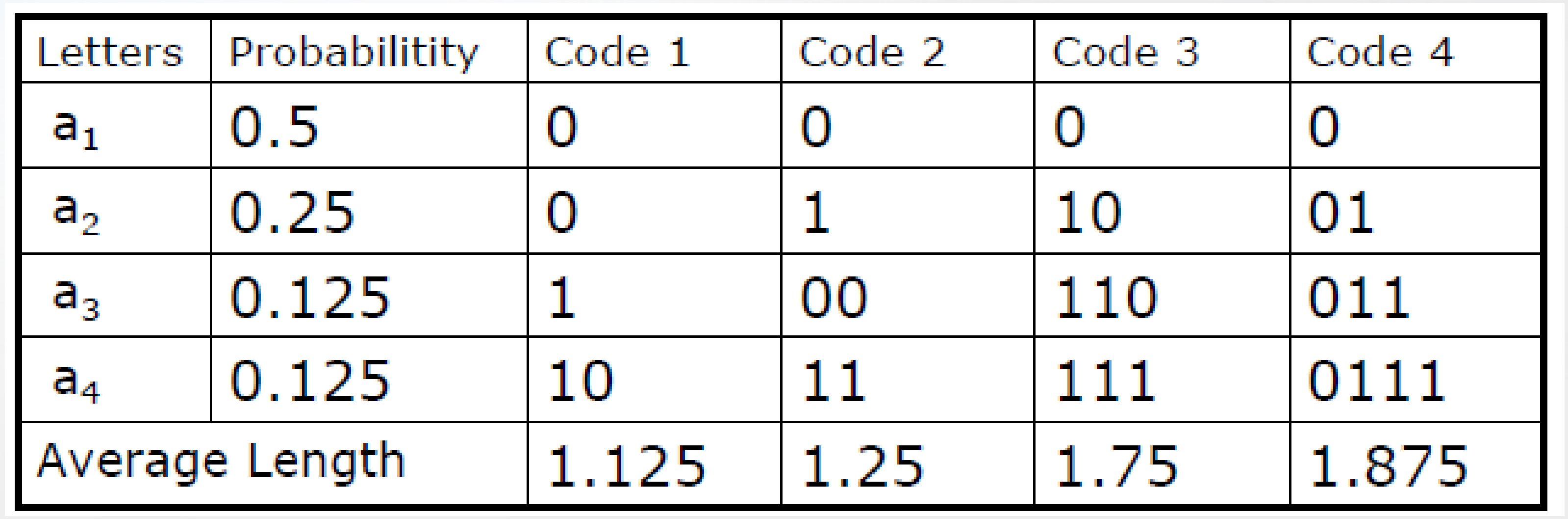
P(a1) = 0.5 P(a2) = 0.25, and P(a3) = P(a4) = 0.125.

- The average length of the code is given by

$$l = \sum_{i=1}^{4} P(a_i) n(a_i)$$

- n(ai) is the number of bits in the codeword for letter ai

Uniquely Decodable Codes



Letters	Probabilitity	Code 1	Code 2	Code 3	Code 4
a_1	0.5	0	0	0	0
a_2	0.25	0	1	10	01
a_3	0.125	1	00	110	011
a_4	0.125	10	11	111	0111
Average Length		1.125	1.25	1.75	1.875

- Code 1: not unique a1 and a2 have the same codeword
- Code 2: not uniquely decodable: 100 could mean a2a3 or a2a1a1
- Codes 3 and 4: uniquely decodable: What are the rules?
- Code 3 is called instantaneous code since the decoder knows the codeword the moment a code is complete.

How do we know a uniquely decodable code?

- Consider two codewords: 011 and 011101
  - Prefix: 011
  - Dangling suffix: 101

- **Algorithm:**
- 1. Construct a list of all the codewords.
- 2. Examine all pairs of codewords to see if any codeword is a prefix of another one. If there exists such a pair, add the dangling suffix to the list unless there is one already.
- 3. Continue this procedure using the larger list until:
  - 1. Either a dangling suffix is a codeword -> not uniquely decodable.
  - 2. There are no more unique dangling suffixes -> uniquely decodable.

Examples of Unique Decodability

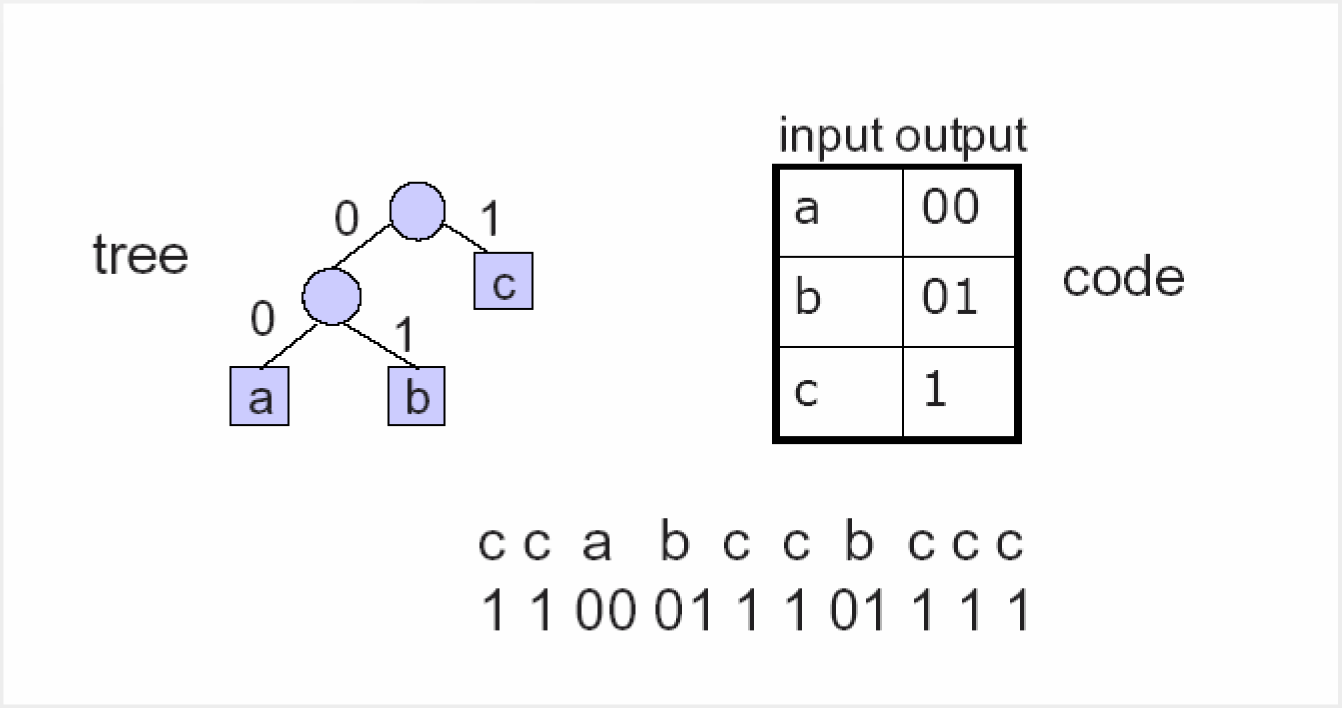
- **Consider {0,01,11}**
  - Dangling suffix is 1 from 0 and 01
  - New list: {0,01,11,1}
  - Dangling suffix is 1 (from 0 and 01, and also 1 and 11), and is already included in previous iteration.
- Since the dangling suffix is not a codeword, {0,01, 11} is uniquely decodable.

Examples of Unique Decodability

- **Consider {0,01,10}**
  - Dangling suffix is 1 from 0 and 01
  - New list: {0,01,10,1}
  - The new dangling suffix is 0 (from 10 and 1).
- Since the dangling suffix 0 is a codeword, {0,01, 10} is not uniquely decodable.

Prefix Codes

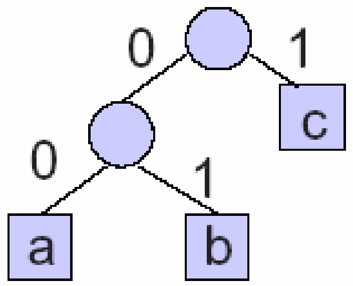
- Prefix codes: A code in which no codeword is a prefix to another codeword.
- A prefix code can be defined by a binary tree
- **Example:**



input	output
a	00
b	01
c	1

```
c c  a   b  c  c  b  c c c
1 1 00 01 1 1 01 1 1 1
```

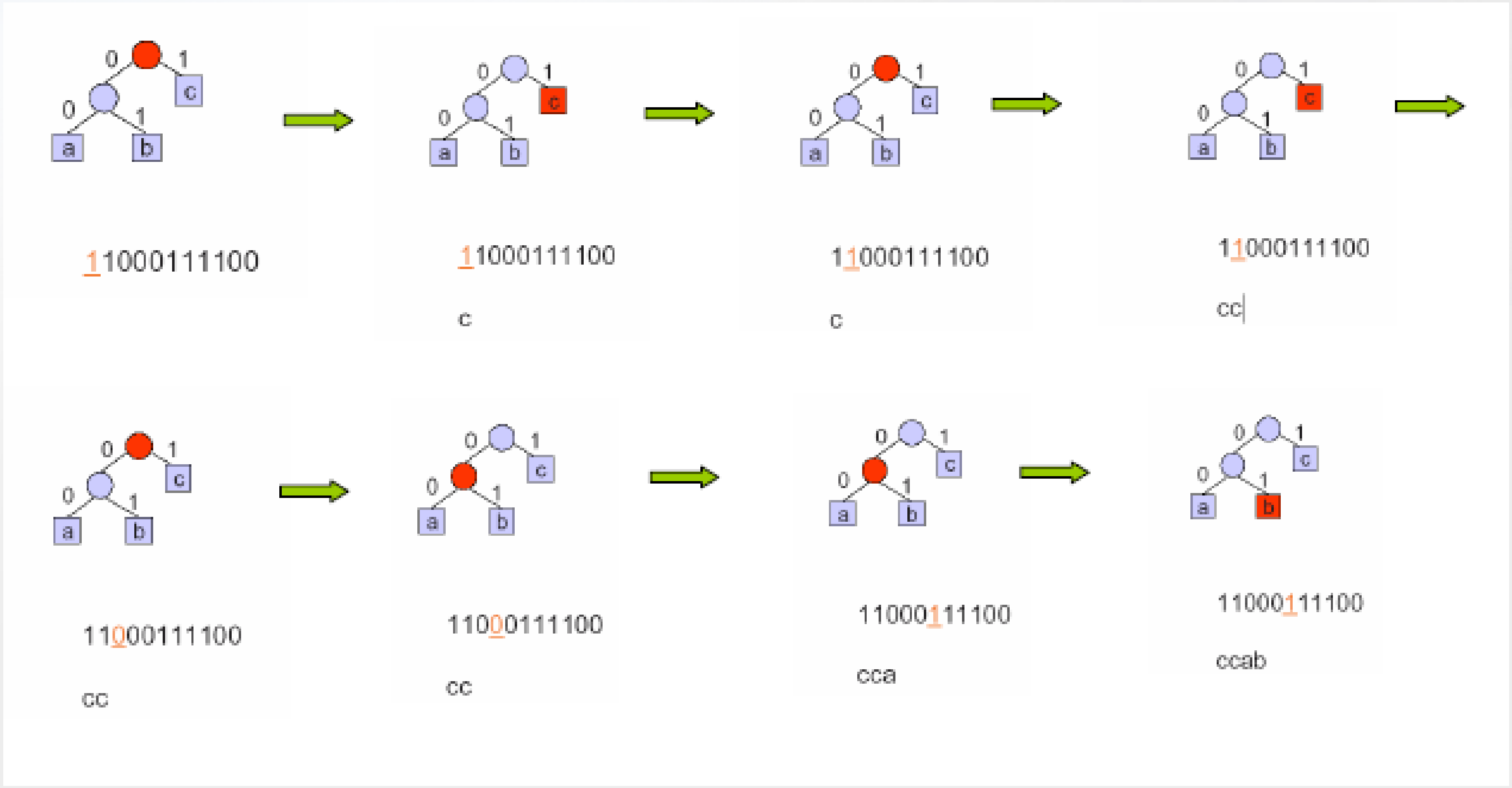
Decoding a Prefix Codeword



```
repeat
start at root of tree
  repeat
    if read bit = 1 then go right
    else go left
  until node is a leaf
  report leaf
until end of the code
```

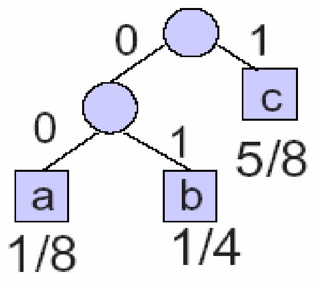
11000111100

Decoding a Prefix Codeword



How good is the code?

- Suppose a, b, and c occur with probabilities 1/8, 1/4, and 5/8, respectively.



bit rate = (1/8)2 + (1/4)2 + (5/8)1 = 11/8 = 1.375 bps
Entropy = 1.3 bps
Standard code = 2 bps

(bps = bits per symbol)