Information, Computation, and Communication

Compression of Data

Objective

- How to **measure the amount of information** present in data?
- How to **store data using the least space possible** (with or without losing information)?

Introduction

- **Why** do you care about **compressing** data?
  - to reduce the storage space used when storing data
  - to reduce transmission time and congestion problems when transmitting data

Introduction

- **French is full of redundancy**!

  Sleon une édtue de l'Uvinertisé de Cmabrigde, l'odrre des ltteers dnas un mot n'a pas d'ipmrotncae, la suele coshe ipmrotnate est que la pmeirère et la drenèire soinet à la bnnoe pclae. Le rsete peut êrte dans un dsérorde ttoal et vuos puoevz tujoruos lrie snas porlbème. C'est prace que le creaveu hmauin ne lit pas chuaqe ltetre elle-mmêe, mias le mot comme un tuot.

- **English as well**!

  Aoccdrnig to a rscheearch at Cmabrigde Uinervtisy, it deosn't mttaer in waht oredr the ltteers in a wrod are, the olny iprmoetnt tihng is taht the frist and lsat ltteer are at the rghit pclae. The rset can be a toatl mses and you can sitll raed it wouthit a porbelm.

- Why is there so much redundancy in languages?

Introduction

- There are two types of compression:

- **Lossless compression**, when you want to keep all the data stored in compressed form.
- Examples: tickets for a concert, tax return, ballot papers, scientific articles

- **Lossy compression**, when not all details are important and a little distortion is allowed.
- Examples: podcasted programs and music tracks in mp3 format, photo sharing on the web, YouTube videos ...

Agenda: This and Next Week

- Notion of entropy
- Lossless compression
- Shannon-Fano algorithm
- Huffman algorithm
- Performance analysis - Shannon's theorem
- Lossy compression

Entropy

Entropy

- Here is a sequence of 16 letters

  A B C D E F G H I J K L M N O P

- Game #1: You have to guess which letter I picked at random by asking a minimum number of questions, to which I can answer only yes or no.

- Solution: 4 questions are needed regardless of the letter that was chosen (cf. binary search algorithm).
  Therefore, we says that the **entropy of this sequence** is 4.
- Note that $16 = 2^4$, so $4 = \log_2(16)$

Entropy

- Here is another sequence of 16 letters (not counting the spaces)

  I L F A I T B E A U A I B I Z A

- Game #2: The game is the same as before.
- **Remarks**:
  - I choose one of the 16 positions by chance (uniformly random).
  - You have to guess only the letter (not the position)

How many binary questions are needed on average to guess the letter?

Entropy

I L F A I T B E A U A I B I Z A

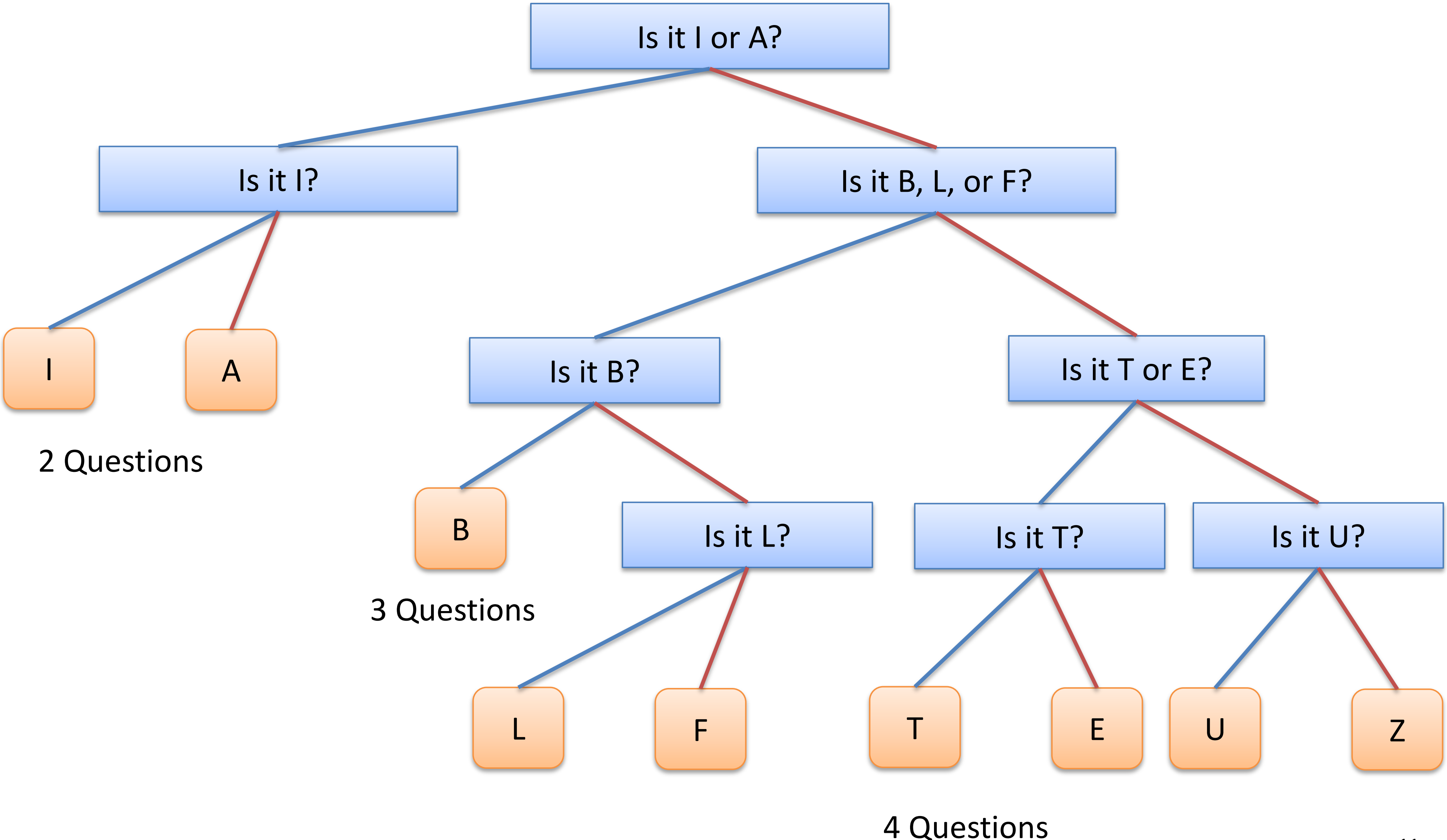
Solution: Sort the letters in descending order of the number of appearances in the sequence:

Letter	I	A	B	L	F	T	E	U	Z
Appearances	4	4	2	1	1	1	1	1	1

The idea is the same as before: we separate the set of letters into two equal parts in terms of number of appearances, which gives:

Question #1: is the letter an I or an A?

- If the answer is yes, **Question #2:** is the letter an I?
- If the answer is no, **Question #2:** is the letter a B, L, or F? etc.


Is it I or A?
Is it I?
Is it B, L, or F?
I
A
2 Questions
Is it B?
Is it T or E?
B
Is it L?
Is it T?
Is it U?
3 Questions
L
F
T
E
U
Z
4 Questions

Entropy

I L F A I T B E A U A I B I Z A

Letter	I	A	B	L	F	T	E	U	Z
Appearances	4	4	2	1	1	1	1	1	1
Questions	2	2	3	4	4	4	4	4	4

Number of questions to ask on average:

$$\frac{4}{16}\cdot 2+\frac{4}{16}\cdot 2+\frac{2}{16}\cdot 3+\frac{1}{16}\cdot 4+\frac{1}{16}\cdot 4+\frac{1}{16}\cdot 4+\frac{1}{16}\cdot 4+\frac{1}{16}\cdot 4+\frac{1}{16}\cdot 4=$$

$$2\cdot\frac{4}{16}\cdot 2+\frac{2}{16}\cdot 3+6\cdot\frac{1}{16}\cdot 4=\frac{16+6+24}{16}=\frac{46}{16}=2.875$$

We say that the **entropy** of this sequence is equal to 2.875.

https://www.shannonentropy.netmark.pl/

Entropy

- Yet another sequence of 16 letters

  A A A A A A A A A A A A A A A A

- Game #3: The game is the same as before

- This time, no question is needed to guess the chosen letter!

- We say that the **entropy of this sequence** is equal to 0.

Entropy

I L F A I T B E A U A I B I Z A

Letter	I	A	B	L	F	T	E	U	Z
Appearances	4	4	2	1	1	1	1	1	1
Questions	2	2	3	4	4	4	4	4	4

Remark:

- To guess a letter that appears 1 time out of 16, we need 4 questions. $4 = \log_2(16)$
- To guess a letter that appears 2 times out of 16 (i.e., p=1/8), we need 3 questions. $3 = \log_2(8)$
- To guess a letter that appears 4 times out of 16 (i.e., p=1/4), we need 2 questions. $2 = \log_2(4)$
- In summary, to guess a letter that appears with a probability of *p*, we need $\log_2(1/p)$ questions.

Entropy

Definition:

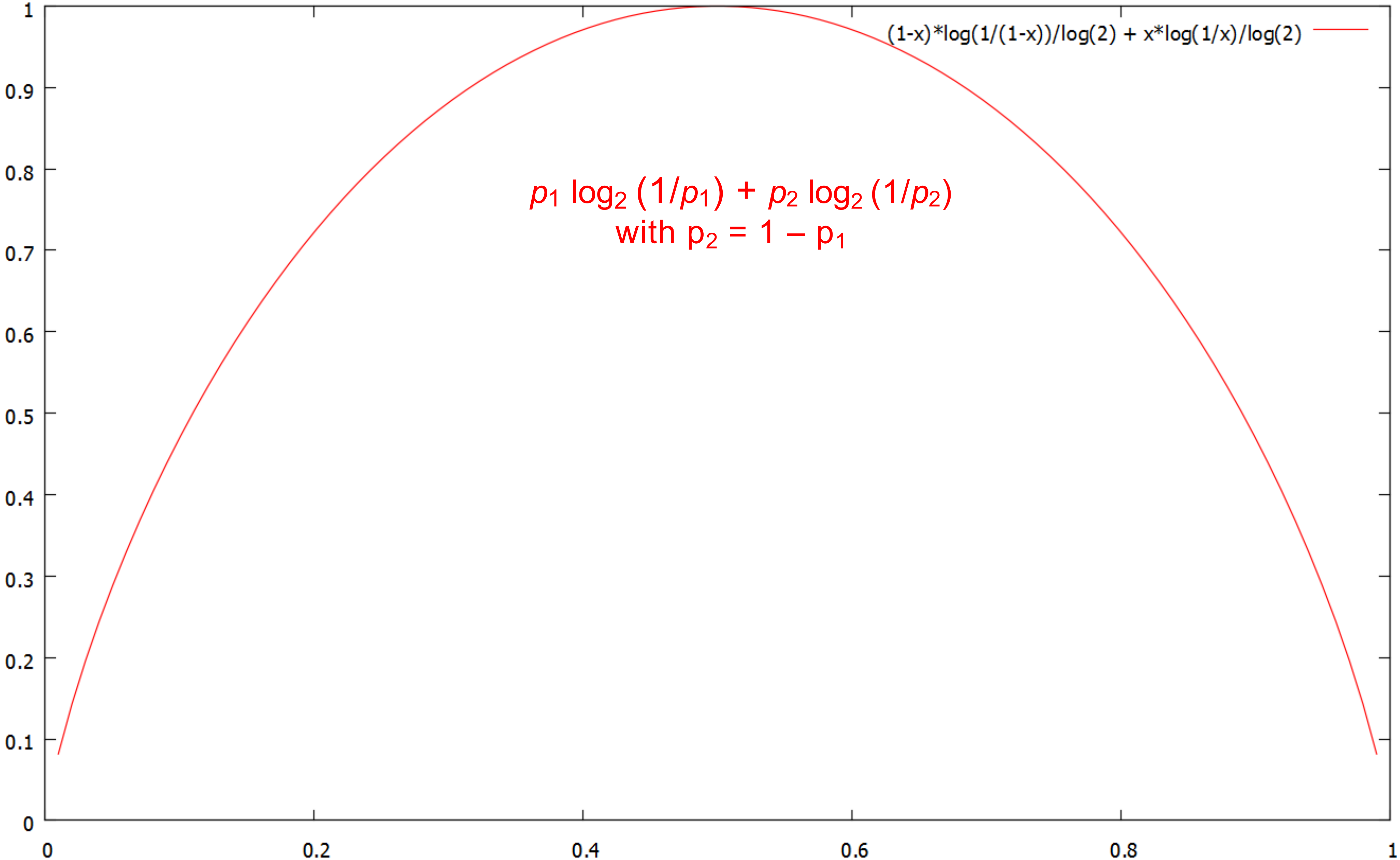
Let X be a sequence of letters from the alphabet $A = \{a_1, \ldots, a_n\}$.

Let p_j be the probability that letter a_j appears in the sequence X.
(Note that $0 \leq p_j \leq 1$ for all j and that $p_1 + \ldots + p_n = 1$).

The **entropy of a sequence** X is defined as

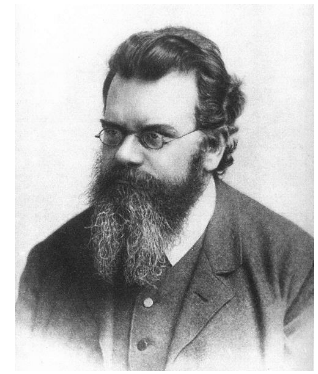
$$H(X) = p_1 \log_2 \left(\frac{1}{p_1}\right) + \ldots + p_n \log_2 \left(\frac{1}{p_n}\right)$$

Therefore, if $p_j = 0$, we set $p_j \log_2 \left(\frac{1}{p_j}\right) = 0$.

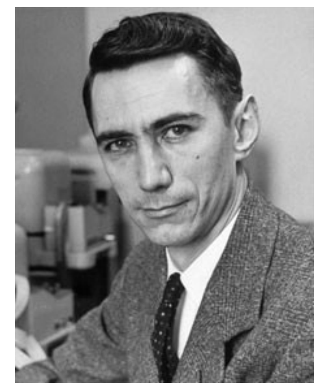

(1-x)*log(1/(1-x))/log(2) + x*log(1/x)/log(2)
$p_1 \log_2 (1/p_1) + p_2 \log_2 (1/p_2)$
with $p_2 = 1 - p_1$
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
0
0.2
0.4
0.6
0.8
1

Entropy

- **Origin in Physics** (Boltzmann, 1872):
  - Entropy **measures disorder** in a physical system.
  - Ludwig Eduard **Boltzmann** (1844-1905)
    - defender of the existence of atoms
    - father of statistical physics



- **Information Theory** (Shannon, 1948):
  - Entropy measures the "**amount of information**" contained in a signal.
  - Claude Edwood **Shannon** (1916-2001)
    - mathematician, electrical engineer, cryptologist,
    - father of information theory, juggler ...



- Or (as we have just seen): Entropy is equal to the average number of questions needed to guess a letter chosen randomly in a sequence.

http://www.science4all.org/article/shannons-information-theory/

Entropy

Interpretation:

M A P J O F B G I L C K D E H N $H(X) = 4$

I L F A I T B E A U A I B I Z A $H(X) = 2.875$

A A A A A A A A A A A A A A A A $H(X) = 0$

The **more different** letters there are in the message, the **more disorder** there is, the **more variety** and therefore the **more "information"** is in the message.

The **more similar** letters there are in the message, the **less disorder** there is, the **more redundancy** there is and therefore the **less "information**” is in the message.

Entropy

- **Interlude**: Which of these two words has the greatest entropy?

E N T R O P I E or D E S O R D R E ?

- Which of these municipalities has the highest/the lowest entropy?

A V E N C H E S, C O S S O N A Y, E C U B L E N S,
G R A N D S O N, L A U S A N N E or M O N T R E U X ?

Entropy

Some properties of entropy:

$$H(X) = p_1 \log_2 \left(\frac{1}{p_1}\right) + \ldots + p_n \log_2 \left(\frac{1}{p_n}\right)$$

- For a given probability of appearance $0 < p \leq 1$, $\log_2(1/p) \geq 0$ but it is not necessarily an integer number (e.g., if $p = 1/3$).

- **Lower bound**: $H(X) \geq 0$ in general and $H(X) = 0$ if and only if the order is total (that is, if all the letters are the same).

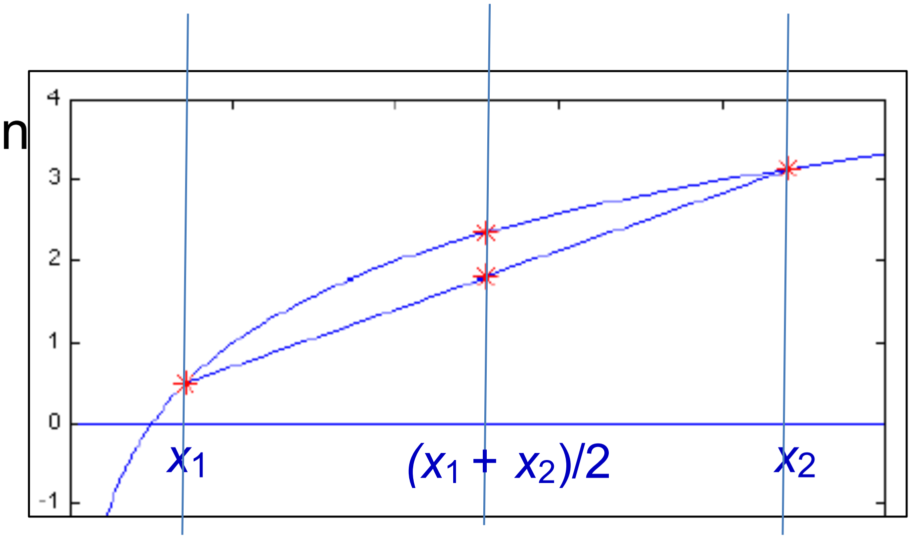
- **Upper bound**: if n is the size of the alphabet, $H(X) \leq \log_2(n)$ in general and $H(X) = \log_2(n)$ if and only if the disorder is total (that is, if all the letters are different).

Lower and Upper Bound (1/2)

$H(X) \geq 0$ in general: note that $0 < p_j \leq 1$ simply implies that $p_j \log_2 (1/pj) \geq 0$, and therefore that $H(X) \geq 0$.

$H(X) \leq \log_2(n)$ in general: note that the function

$f(x) = \log_2(x)$ is concave for $x \geq 0$:



In particular, this means that:

$$\frac{\log_2(x_1) + \log_2(x_2)}{2} \leq \log_2 \left(\frac{x_1 + x_2}{2}\right) \quad \text{for all } x_1, x_2 \geq 0$$

More generally, if $0 \leq p_1, p_2 \leq 1$ and $p_1 + p_2 = 1$, then:

$$p_1 \log_2(x_1) + p_2 \log_2(x_2) \leq \log_2(p_1 x_1 + p_2 x_2) \text{ for all } x_1, x_2 \geq 0$$

Lower and Upper Bound (2/2)

Even more generally, if $0 \leq p_j \leq 1$ et $p_1 + \ldots + p_n = 1$, then

$$p_1 \log_2(x_1) + \ldots + p_n \log_2(x_n) \leq \log_2(p_1 x_1 + \ldots + p_n x_n) \text{ for all } x_1, \ldots, x_n \geq 0.$$

Applying this inequality with $x_j = 1/p_j$ we finally get

$$\begin{aligned} H(X) &= p_1 \log_2\left(\frac{1}{p_1}\right) + \ldots + p_n \log_2\left(\frac{1}{p_n}\right) \\ &\leq \log_2\left(\frac{p_1}{p_1} + \ldots + \frac{p_n}{p_n}\right) = \log_2(1 + \ldots + 1) = \log_2(n) \end{aligned}$$

Application of Entropy: Lossless Compression

Lossless Compression

- As discussed, data compression allows us to
  - **reduce the storage** used to store data
  - **reduce transmission time** and congestion problems when transmitting data
- The basic principle behind data compression is to **suppress redundancy** contained in the data (e.g., letters or words that often come up in a message are abbreviated)

- When talking about **lossless compression**, it is required that the original message can be completely recovered from the compressed data.

Lossless Compression

- Examples of lossless compression algorithms
  - **Language in txt-messages** (SMS): "slt" (salut), "tqt" (t'inquiète), "thx" (thanks), "lol" (laughing out loud), etc. words that are used often are reduced to short sequences of letters.
  - **Acronyms**: EPFL, UNIL, SV, IC, etc...
  - **Morse Code**: "e" = .   "a" = .–   "s" = ...   "t" = –
    but "x"= –..–   "z"= – – ..

- The concept of **entropy** makes it possible to compress data in a **systematic and optimal way**.

Lossless Compression

- Let's go back to our examples: suppose you want to send a txt (SMS) with the following message to a friend:

I L F A I T B E A U A I B I Z A

- The goal of the game is now to minimize the number of bits needed to represent this message.

Important notes:

- The person who receives your message must be able to read it!
- Before sending the message, the sender and the recipient can (and even have to) agree on a **common dictionary**.
  Example: A = '01', B = '10', etc.

Lossless Compression

I L   F A I T   B E A U   A   I B I Z A

- **Solution 1**: use the ASCII code (see slides of lesson “Information Representation): each letter is represented by 8 bits; 16 × 8 = 128 bits are needed to represent this message.

- **Solution 2**: Note that the message to be represented is composed of only 16 letters: we only need 4 bits per letter ($2^4 = 16$); and therefore 16 × 4 = 64 bits in total.

- **Solution 3**: Notice that some letters are repeated: there are actually only 9 different letters. And some are more common than others ...
- How to proceed?

Shannon-Fano Algorithm

- Robert Fano (1917-2016)
  - professor at MIT
  - another pioneer of information theory

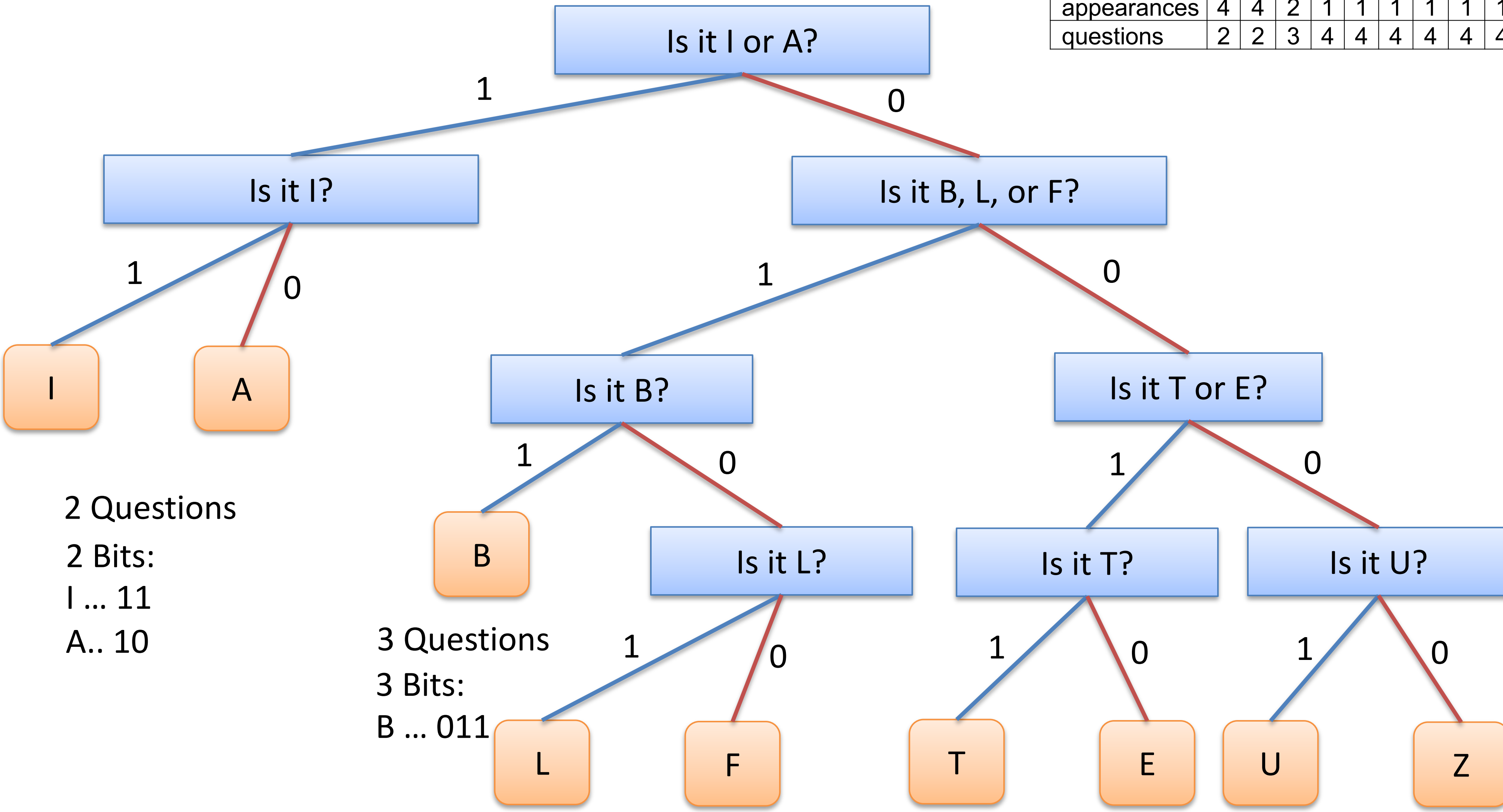
- Let's go back to our questions for "I L F A I T B E A U A I B I Z A"

letter	I	A	B	L	F	T	E	U	Z
appearances	4	4	2	1	1	1	1	1	1
questions	2	2	3	4	4	4	4	4	4

- In order to assign a bit sequence (a "code word") to each letter, we follow the following two rules:
  - **Rule 1**: The number of bits attributed to each letter is equal to the number of questions needed to guess it.
  - **Rule 2**: Bits 0 or 1 are assigned based on the answers to the questions. More precisely, the bit number j is equal to 1 or 0 depending on whether the answer to the question was yes or no.

I L F A I T B E A U A I B I Z A

letter	I	A	B	L	F	T	E	U	Z
appearances	4	4	2	1	1	1	1	1	1
questions	2	2	3	4	4	4	4	4	4



2 Questions
2 Bits:
I … 11
A.. 10

3 Questions
3 Bits:
B … 011

4 Questions
4 Bits: L … 0101, F…0100, ….

Shannon-Fano Algorithm

letter	I	A	B	L	F	T	E	U	Z
code word	11	10	011	0101	0100	0011	0010	0001	0000

- The message "IL FAIT BEAU A IBIZA" is written as follows:

11 0101 0100 10 11 0011 . . .

- **Observation 1**: In the dictionary, no code word is the **prefix of another**. Therefore, the encoded message can be decoded letter by letter (without waiting for the next letter or the end of the message).

Shannon-Fano Algorithm

letter	I	A	B	L	F	T	E	U	Z
appearances	4	4	2	1	1	1	1	1	1
code word	11	10	011	0101	0100	0011	0010	0001	0000

- **Observation 2**: The number of used bits is equal to

$$8 \times 2 + 2 \times 3 + 6 \times 4 = 16 + 6 + 24 = 46$$

- Compared to Solution 2, which requires 64 bits, one saves approximately 25% of bits.
- Since the message has 16 letters, the average number of bits used per letter is $46/16 = 2.875 = H(X)$ = **entropy of the message**

- We will see later that the **entropy is a lower bound** on the average number of bits per letter that are required: no compression algorithm can go below this bound.
- This is the statement of one of **Shannon's theorems**.

Shannon-Fano Algorithm (Continued)

- Let’s consider another sequence (12 letters in total, without spaces):

JE PARS A PARIS

- How would you proceed to represent this sequence with a minimum number of bits per letter?
- Let's go back again to our questions game and build the same table as before, sorting the letters in descending order of the number of appearances:

letter	A	P	R	S	J	E	I
appearances	3	2	2	2	1	1	1

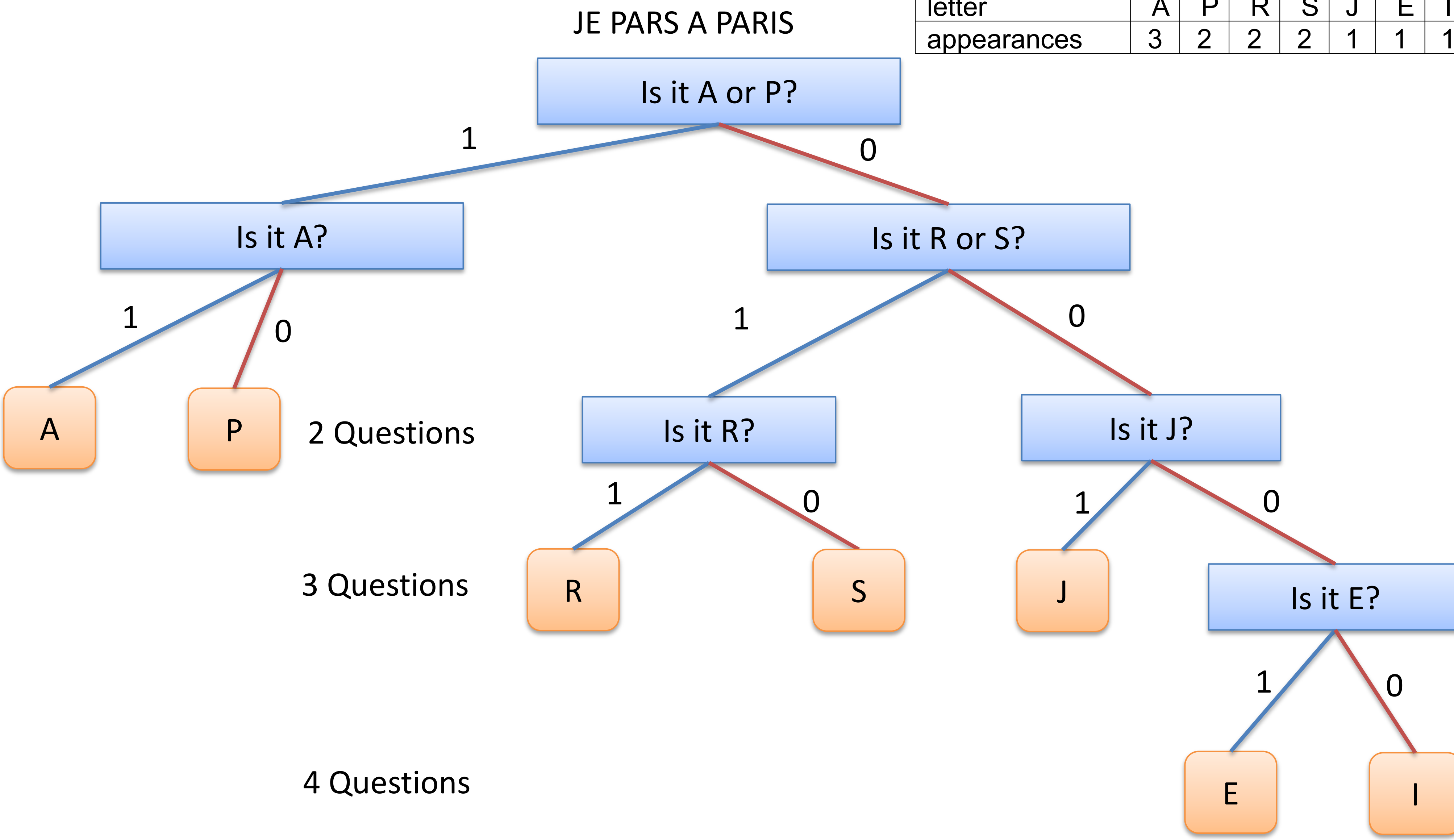
- Note here that it is **not possible to divide** the set of letters into two parts such that the number of appearances on the left **equals** the number of appearances on the right.

Shannon-Fano Algorithm (Continued)

letter	A	P	R	S	J	E	I
appearances	3	2	2	2	1	1	1

- However, we make sure to **minimize the difference** between the number of appearances on the left and on the right.
- In the example above, there are actually two possibilities for the first question:
  - Is the letter an A or a P? (5 appearances left, 7 right)
  - Is the letter an A, P or R? (7 left, 5 right)
- Suppose we choose the first possibility; the game continues as before, with this new a little **more flexible rule**...

letter	A	P	R	S	J	E	I
appearances	3	2	2	2	1	1	1



Shannon-Fano Algorithm (Continued)

- Which brings us, e.g., to the following table:

letter	A	P	R	S	J	E	I
appearances	3	2	2	2	1	1	1
questions	2	2	3	3	3	4	4

- The average number of questions needed to guess a letter is:

$$\frac{5}{12}\cdot 2+\frac{5}{12}\cdot 3+\frac{2}{12}\cdot 4=\frac{10+15+8}{12}=\frac{33}{12}=2.75$$

- The rule of assigning code words to each letter is exactly the same as before:
  - **Rule 1**: the number of bits attributed to each letter is equal to the number of questions necessary to guess it.
  - **Rule 2**: Bits 0 or 1 are assigned based on the answers to the questions. More precisely, the bit number j is equal to 1 or 0 depending on whether the answer to the question was yes or no.

Shannon-Fano Algorithm (Continued)

- This gives the following dictionary:

letter	A	P	R	S	J	E	I
appearances	3	2	2	2	1	1	1
questions	2	2	3	3	3	4	4
code word	11	10	011	010	001	0001	0000

(Recall: by construction no code word is the prefix of another)

- The number of bits necessary to represent this sequence is

$$5 \cdot 2 + 5 \cdot 3 + 2 \cdot 4 = 33$$

- Therefore, the average number of bits used to represent a letter in the sequence “JE PARS A PARIS” is again

$$\frac{33}{12} = 2.75$$

Shannon-Fano Algorithm (Continued)

- Now, let's calculate the entropy of this sequence (with 12 letters):

letter	A	P	R	S	J	E	I
appearances	3	2	2	2	1	1	1
probability	1/4	1/6	1/6	1/6	1/12	1/12	1/12

(Recall that $\log_2(1/p)$ is not necessarily an integer)

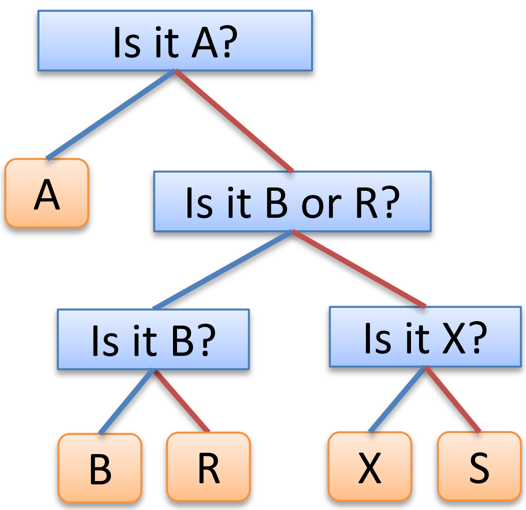
- The calculation of the entropy gives

$$H(X) = \frac{1}{4}\log_2(4) + 3 \cdot \frac{1}{6}\log_2(6) + 3 \cdot \frac{1}{12}\log_2(12) = 2.69 \text{ bit}$$

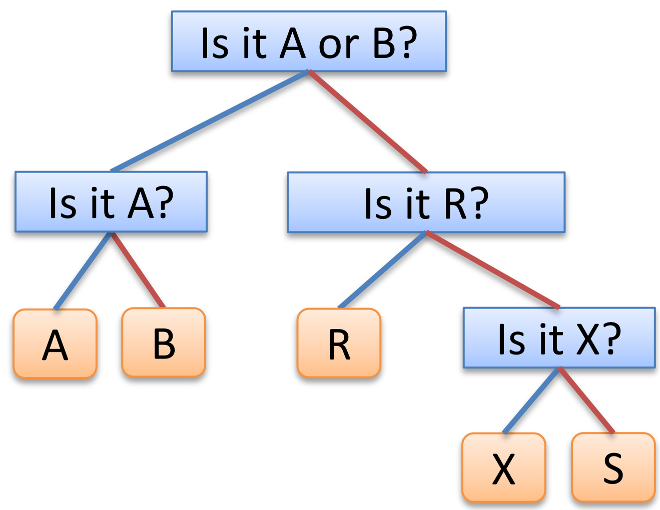
- which is a bit smaller than the value (2.75) that we computed on the previous slide.

Shannon-Fano Algorithm (Continued)

- **Observation**: from the point of view of the compression ratio (average bit number per letter) the Shannon-Fano algorithm offers a **very good performance** in the majority of cases but is **not guaranteed to be optimal**.
- The compression ratio may be **different depending on the grouping**, even when the rule of minimizing the difference between the groups is respected.
- **Example**: let's consider the sequence ABRAXAS (3x A + 1x {B,R,X,S})

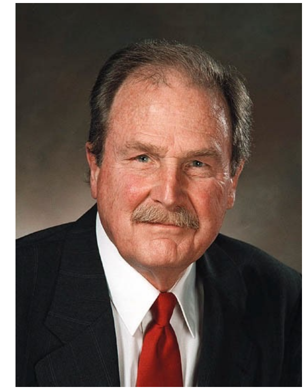


Total: $3 \cdot 1 + 4 \cdot 3 = 15$ bit



Total: $(3+1+1) \cdot 2 + 2 \cdot 3 = 16$ bit

Huffman



David Albert Huffman
(1925 – 1999)
Professor at MIT and
UC Santa Cruz

- In practice, we use the **Huffman algorithm,** which follows a **bottom-up approach** to grouping the number of appearances

  - The algorithm works with **groups of letters** and their total **number of appearances**
  - Initially all letters are in their own group labeled with the number of appearances of the corresponding letter.
  - Then, **while there is more than one group** left, the algorithm applies the following steps:
    1. Choose **two groups** with the **smallest number of appearances**
    2. Replace these two groups with a **new group** that **merges the letters** and **adds the number of appearances** of the two chosen groups.

- The Huffman algorithm always achieves the **optimal compression ratio.**

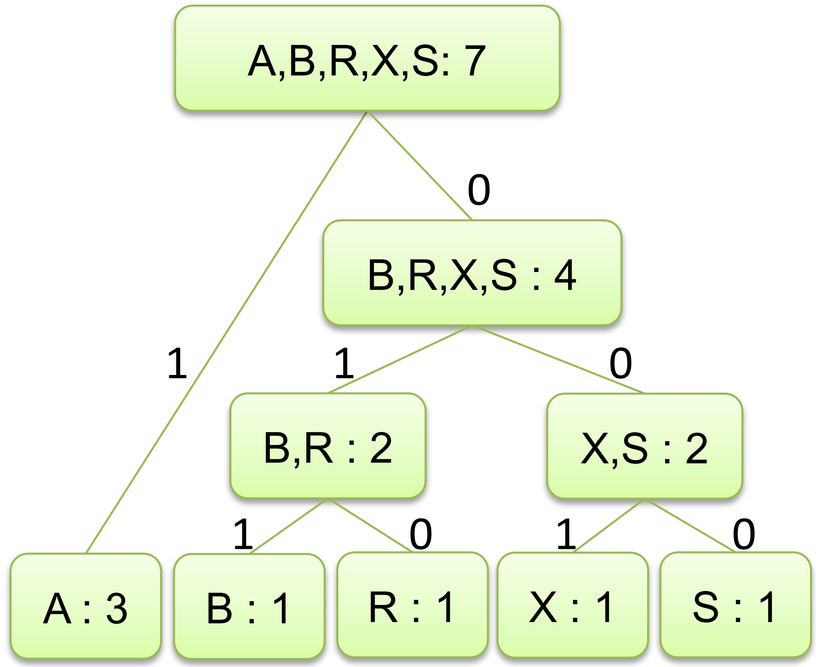
Exercise: Apply Huffman's algorithm to ABRAXAS

letter	A	B	R	X	S
appearances	3	1	1	1	1

Exercise: Apply Huffman's algorithm to ABRAXAS

Letter	Code	Length	App.
A	1	1	3
B	011	3	1
R	010	3	1
X	001	3	1
S	000	3	1

$$3 \cdot 1 + (1 + 1 + 1 + 1) \cdot 3 = 15$$



letter	A	B	R	X	S
appearances	3	1	1	1	1

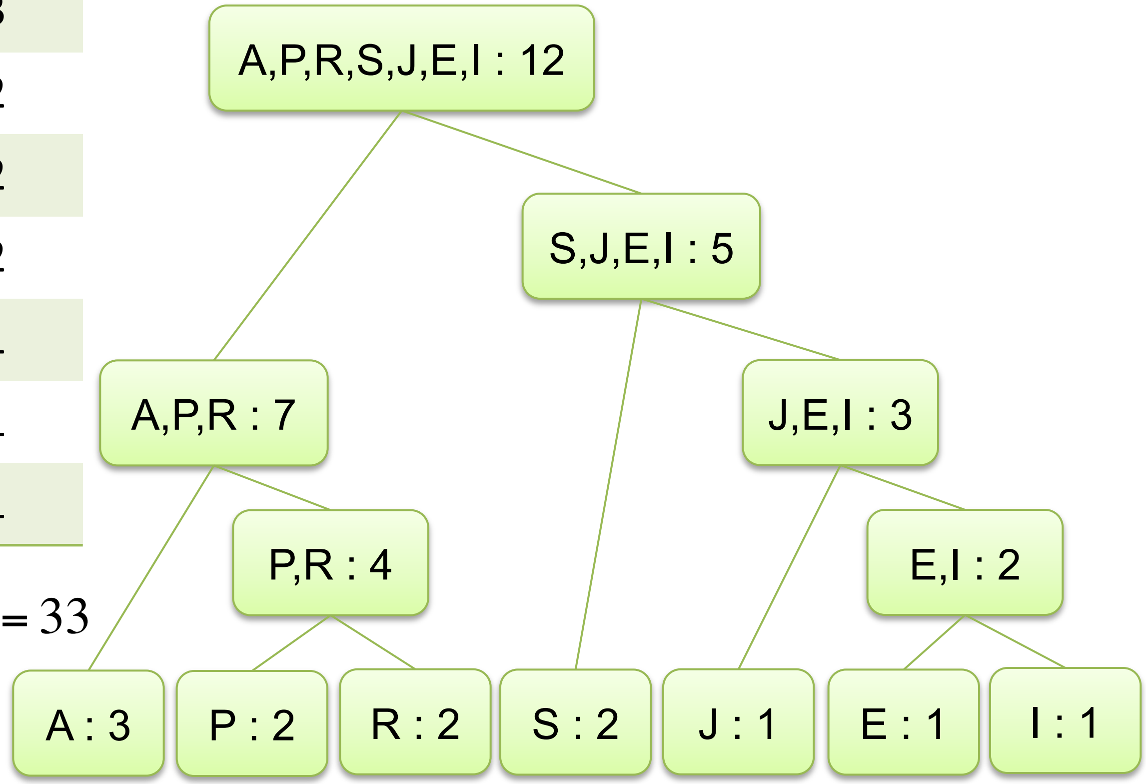
Exercise: Apply Huffman's algorithm to JE PARS A PARIS

letter	A	P	R	S	J	E	I
appearances	3	2	2	2	1	1	1

Exercise: Apply Huffman's algorithm to JE PARS A PARIS

Letter	Code	Length	App.
A	11	2	3
P	101	3	2
R	100	3	2
S	01	2	2
J	001	3	1
E	0001	4	1
I	0000	4	1

$$(3+2)\cdot 2+(2+2+1)\cdot 3+2\cdot 4=33$$



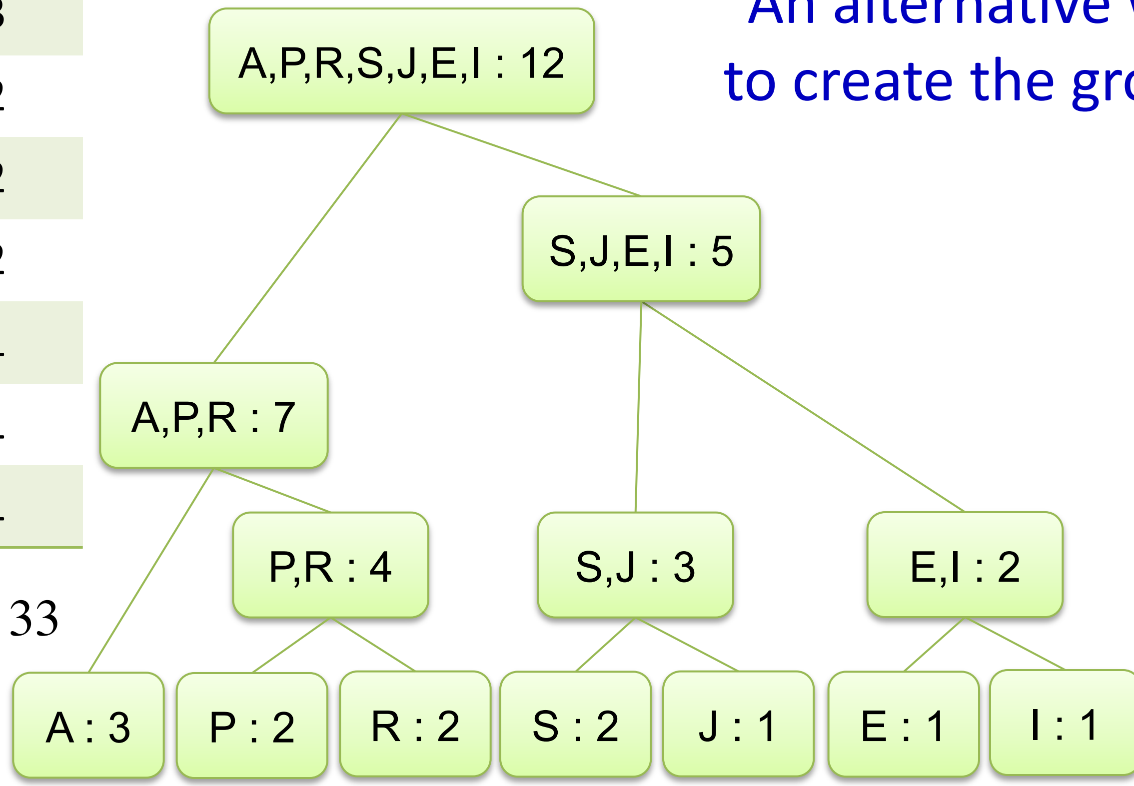
letter	A	P	R	S	J	E	I
appearances	3	2	2	2	1	1	1

Exercise: Apply Huffman's algorithm to JE PARS A PARIS

Letter	Code	Length	App.
A	11	2	3
P	101	3	2
R	100	3	2
S	011	3	2
J	010	3	1
E	001	3	1
I	000	3	1

$$3 \cdot 2 + (2+2+2+1+1+1) \cdot 3 = 33$$

An alternative way to create the groups!



letter	A	P	R	S	J	E	I
appearances	3	2	2	2	1	1	1

Applications of Huffman's Algorithm

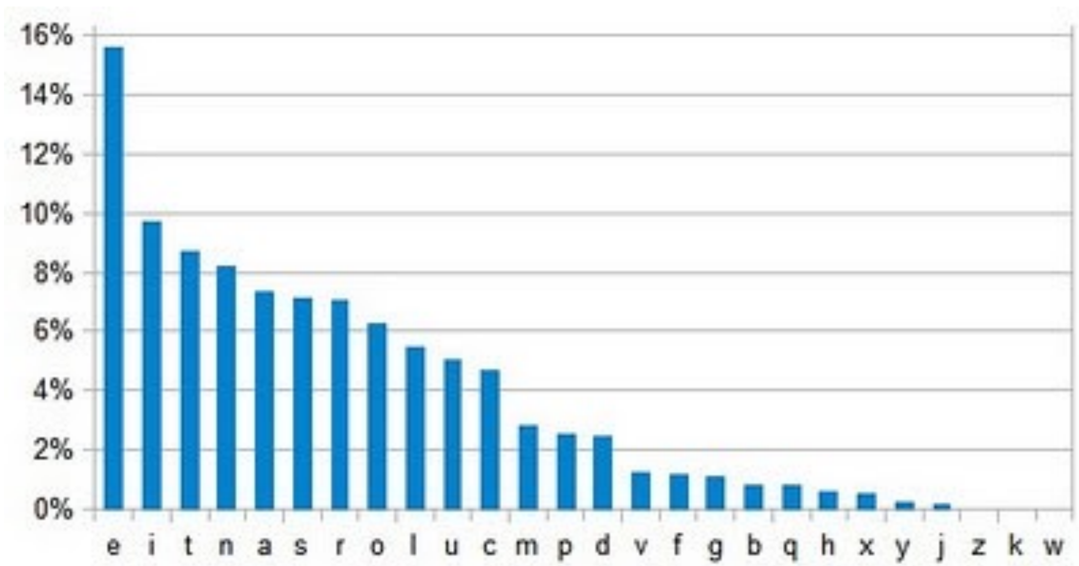
With this algorithm, we obtain the following reductions:

- For a text file:
  - 15-25% reduction when using letters
  - Up to 75% reduction when using words instead of letters

- For a data file: 25-30% reduction (e.g., in zip files aka "compressed folder" in Windows)

More about Encodings ...

- **Question**: Should we always create an **encoding "from scratch"** to represent a message?
- **Answer**: Not necessarily! One can use a code based on the **probabilities of appearance** of letters in the French language:



(Taken from the Universal Declaration of Human Rights)

Optimal Encoding?

- This is a **compromise**: an encoding can be optimal for the distribution of appearance of letters for a given language but not be for a particular text.
- The example below comes from a 300-page novel that respects a constraint visible in this excerpt:

Il poussa un profond soupir, s'assit dans son lit, s'appuyant sur son polochon. Il prit un roman, il l'ouvrit, il lut; mais il n'y saisissait qu'un imbroglio confus, il butait à tout instant sur un mot dont il ignorait la signification.

(excerpt from "La disparition" by Georges Perec, 1969)

Performance Analysis

Performance Analysis: Definition (1/2)

- A **binary encoding (or code)** is a set C of elements $c_1, \ldots, c_n$ (also called code words) that are sequences of 0 and 1 of finite length.
  Example: { 11, 10, 011, 010, 001, 0001, 0000 }
- We use l_j to denote the length of the code word c_j.
  Example: $l_5 = 3$
- A binary encoding is **prefix-free** if no code word is the prefix of another code word. This guarantees
  - that **all code words are different**;
  - that any message formed of these code words can be **decoded while reading**.
  - Example: with the encoding above, 1101010011 reads:
    11 010 10 011
- A prefix-free encoding is also called **prefix code**.

Performance Analysis: Definitions (2/2)

- The encoding $C = \{c_1, \ldots, c_n\}$ can be used to represent a sequence X formed with letters from an alphabet $A = \{a_1, \ldots, a_n\}$: each letter a_j is represented by the code word c_j of length l_j.
- Example:

letter	A	P	R	S	J	E	I
code word	11	10	011	010	001	0001	0000

- If the letters $a_1, \ldots, a_n$ appear with probabilities $p_1, \ldots, p_n$ in the sequence X, then the **average code length** $L(C)$,i.e., the average number of bits used per letter, is given by

$$L(C) = p_1\, l_1 + \ldots + p_n\, l_n$$

Shannon's Theorem

- For every prefix-free binary encoding C that is used to represent a given sequence X, the following inequality is true:

$$H(X) \leq L(C)$$

- Note that this is again a **threshold** (like in the Sampling Theorem): below this threshold, it is **not possible** to compress data **without losing information.**

- In order to prove Shannon's theorem, we need the following inequality.

Kraft Inequality

- Let $C = \{c_1, . . . , c_n\}$ a prefix-free binary encoding.
  Then the following inequality is true.

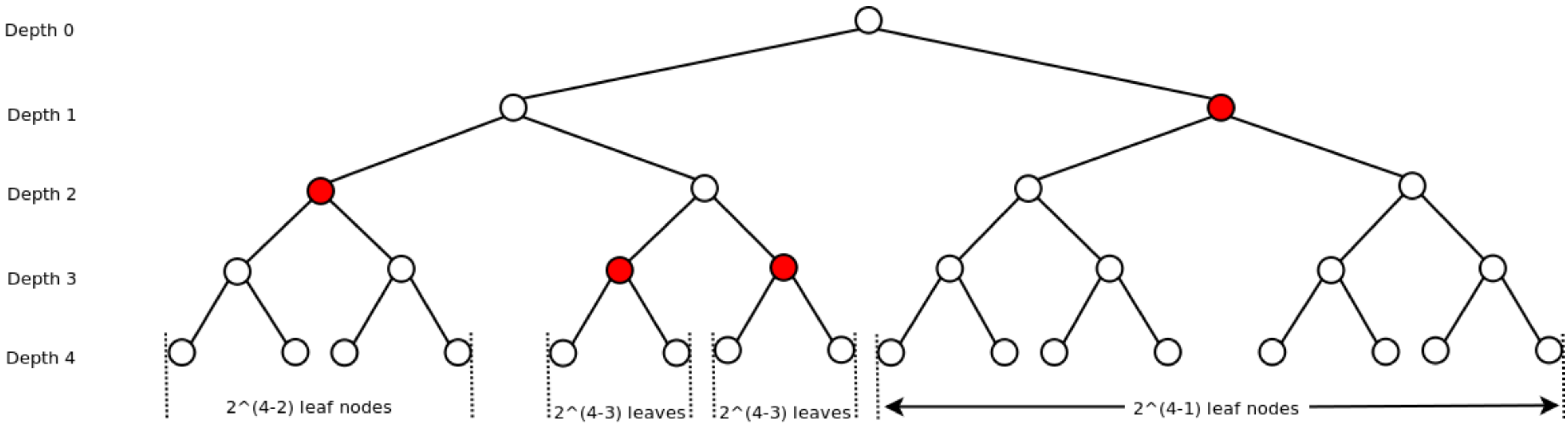
$$2^{-l_1} + \ldots + 2^{-l_n} \leq 1$$

(where l_j is the length of the code word c_j)

Proof:

- Let $l_{\max}$ be the length of the longest code word in C.
- Each code word of C can be represented by a node in a binary tree of depth $l_{\max}$.
- We call the set of all nodes below a code word c_i in this tree, the descendants of c_i.

Kraft Inequality



- At depth $l_{\max}$ there are $2^{l_{\max}}$ leaf nodes.
- The code word c_j has $2^{(l_{\max} - l_j)}$ descendants at depth $l_{\max}$.
- Since the encoding is prefix-free, the descendants of distinct code words are all distinct. Therefore, we know that

$$2^{l_{\max}-l_1} + \ldots + 2^{l_{\max}-l_n} \leq 2^{l_{\max}}$$

And after dividing by $2^{l_{\max}}$ we get $2^{-l_1} + \ldots + 2^{-l_n} \leq 1$

Shannon's Theorem: Proof (1/2)

$H(X) \leq L(C)$

$H(X) - L(C) \leq 0$

- By definition:

$$H(X) - L(C) = \left(p_1 \log_2\left(\frac{1}{p_1}\right) + \ldots + p_n \log_2\left(\frac{1}{p_n}\right)\right) - (p_1 \ell_1 + \ldots + p_n \ell_n)$$

$$= p_1 \left(\log_2\left(\frac{1}{p_1}\right) - \ell_1\right) + \ldots + p_n \left(\log_2\left(\frac{1}{p_n}\right) - \ell_n\right)$$

- Since $-l_j = \log_2(2^{-l_j})$ and $\log_2(a) + \log_2(b) = \log_2(a \cdot b)$ we know that

$$\log_2\left(\frac{1}{p_j}\right) - \ell_j = \log_2\left(\frac{1}{p_j}\right) + \log_2\left(2^{-\ell_j}\right) = \log_2\left(\frac{2^{-\ell_j}}{p_j}\right)$$

- And therefore:

$$H(X) - L(C) = p_1 \log_2\left(\frac{2^{-\ell_1}}{p_1}\right) + \ldots + p_n \log_2\left(\frac{2^{-\ell_n}}{p_n}\right)$$

Shannon's Theorem: Proof (2/2)

- Using the fact that $f(x) = \log_2(x)$ is concave, we obtain

$$H(X) - L(C) = p_1 \log_2\left(\frac{2^{-\ell_1}}{p_1}\right) + \ldots + p_n \log_2\left(\frac{2^{-\ell_n}}{p_n}\right)$$

$$\leq \log_2\left(p_1 \frac{2^{-\ell_1}}{p_1} + \ldots + p_n \frac{2^{-\ell_n}}{p_n}\right) = \log_2\left(2^{-\ell_1} + \ldots + 2^{-\ell_n}\right)$$

- Finally, with the Kraft inequality and the fact that $f(x) = \log_2(x)$ is increasing, we conclude that

$$H(x) - L(C) \leq \log_2(1) = 0$$

$$H(X) \leq L(C)$$

Performance of Shannon-Fano and Huffman

- We can also show that with the Shannon-Fano algorithm,

$$L(C) \leq H(X) + 1 \quad \text{(bit /lettre)}$$

- So, its performance is close to the best performance we can hope for.

$$H(X) \leq L(C) \leq H(X) + 1 \quad \text{(bit /lettre)}$$

- **Remarks**:
  - The above inequality is generally pessimistic. We have seen in the preceding example that $L(C)$ can be closer to $H(X)$, and even equal to $H(X)$ in the ideal case.
  - The proof of this inequality is a little technical, so we will not do it here.
  - The performance of the Huffman algorithm (which is **optimal**) is also within these limits.
    (Optimality proof, e.g., http://www.cs.utoronto.ca/~brudno/csc373w09/huffman.pdf)

Summary

- Entropy measures the "**amount of information**" present in a message.
- For every **prefix code** C that is used to represent a given sequence X, the **average code length** $L(C)$,i.e., the average number of bits used per letter, **cannot be lower than the entropy**, i.e., $H(X) \leq L(C)$, i.e., it is **not possible** to compress with an average code length that is lower than $H(X)$ **without losing information.**
- The average code length $L(C)$ in a message that was encoded with the **Shannon-Fano's algorithm** is

$$H(X) \leq L(C) \leq H(X)+1$$

- Huffman's algorithm is a bottom-up version of Shannon-Fano.
  It generates the **optimal prefix code**.