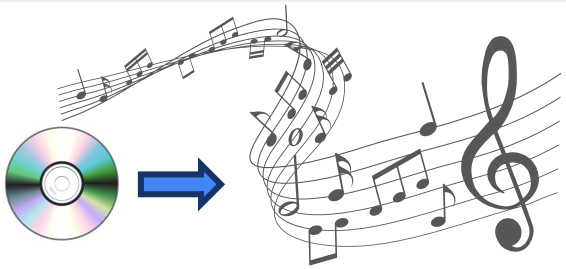
Information, Calcul et Communication

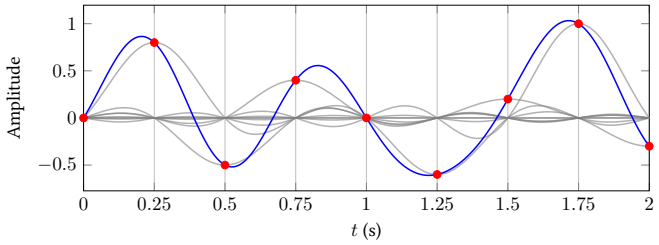
CS-119(g) ICC – Théorie
Semaine 6

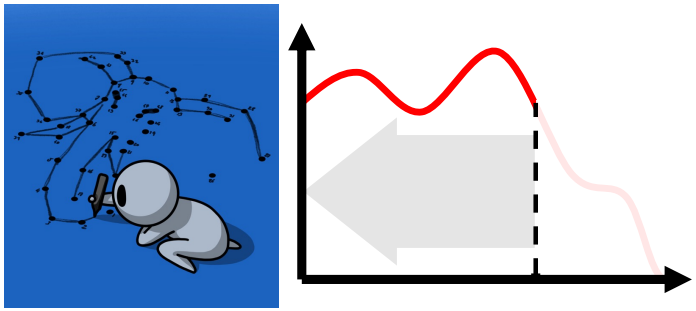
Rafael Pires
rafael.pires@epfl.ch

Lausanne, 17.10.2025

Précédemment, dans ICC-T 5







- **Réconstruction de signaux** (interpolation optimale)

$$X_I(t) = \sum_{m \in \mathbb{Z}} X(mT_e) \cdot \mathrm{sinc}\left(\frac{t - mT_e}{T_e}\right)$$

- **Théorème d'échantillonnage** (Nyquist–Shannon)
  - Condition de reconstruction :

$$f_e > 2f_{max}$$

  - Effet stroboscopique quand $f_e < 2f_{max}$
    → perte d'information

- **Filtrer avant échantillonner** (anti-aliasing)
  - Filtre passe-bas idéal $f_c < {}^{f_e}/_2$ avant l'échantillonnage

Programme du cours

Calcul

- Algorithmes
- Complexité
- Conception d'algorithmes
- Calculabilité
- **Circuits, architecture**

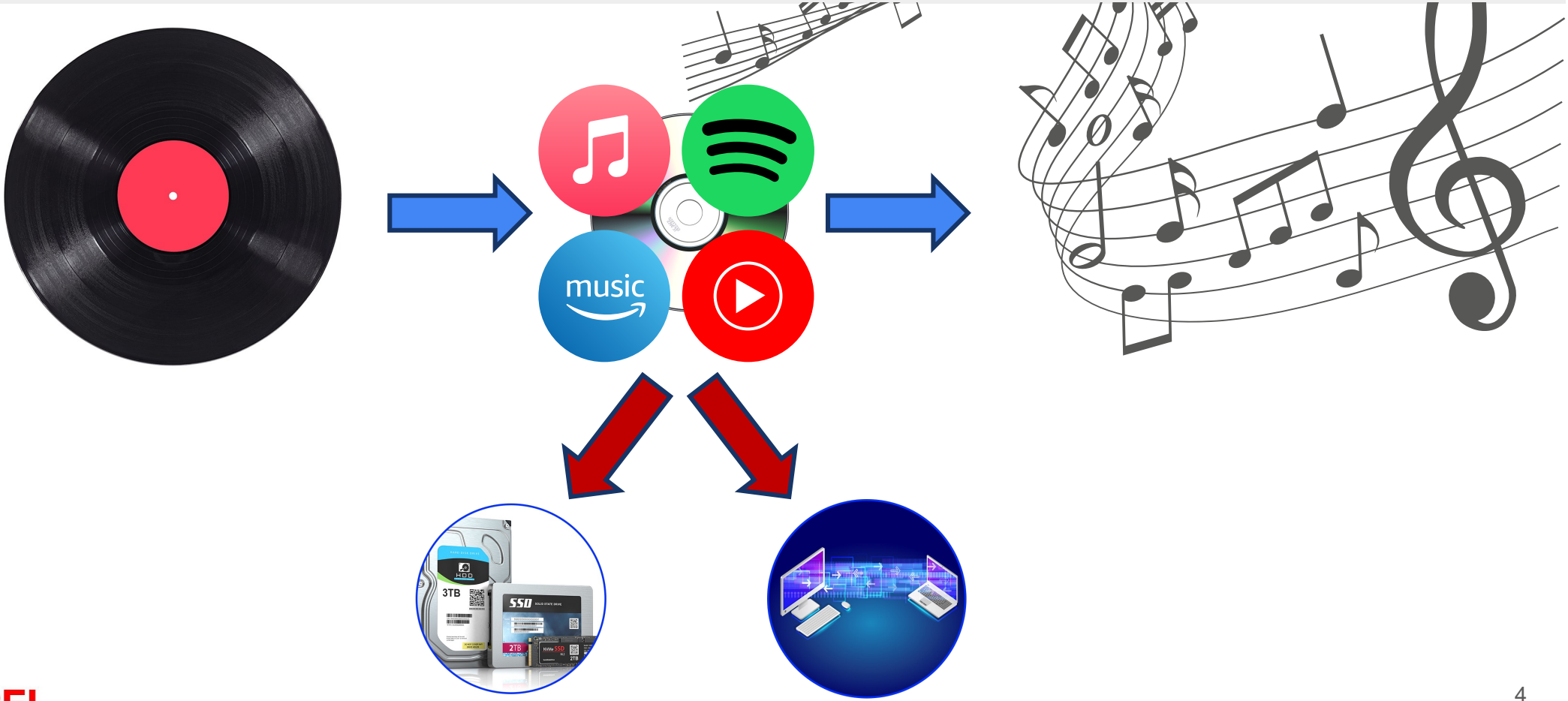
Information

- **Représentation de nombres**
- **Echantillonnage et reconstruction de signaux**
- Entropie
- Compression

Communication

- **Réseau**
- **Cryptographie**

Compression des données



Aujourd'hui

- Compression des données
- Entropie
- Algorithmes de Shannon-Fano et Huffman
- Théorème de Shannon
- Compression avec pertes

Pourquoi donc vouloir compresser des données ?

Deux raisons principales :

- Pour réduire l'espace utilisé lors du **stockage** de ces données
- Pour réduire le temps de **transmission** de ces données

Quels types de données peuvent être compressées ?

- Les textes
- Les sons
- Les images
- Les vidéos
- En général, tout type de données numériques !

Deux types de compression

- **Compression sans pertes :** lorsqu'on désire retrouver l'intégralité des données stockées sous forme compressée
  - **Exemples :** billets pour un concert, bulletins de vote, articles scientifiques

- **Compression avec pertes :** lorsqu'on n'est pas tant à cheval que ça sur les détails et qu'on s'autorise un peu de distorsion
  - **Exemples :** morceaux de musique au format mp3, partage de photos sur le web, vidéos YouTube...

Compression sans pertes

```
print("bla bla bla bla bla ")
print(5*"bla ")
```

- Aussi étrange que cela puisse paraître, il est possible de réduire la taille d'un fichier informatique…
- …sans pour autant perdre la moindre information à propos du fichier !
- L'idée de base consiste à **supprimer/réduire la redondance** présente dans les données en **abrégeant les motifs qui reviennent souvent** dans celles-ci.

La langue française est pleine de redondance !

```
print("bla bla bla bla bla ")
print(5*"bla ")
```

- Pour preuve, voici deux phrases :

~~Cette première phrase dont les lettres ont été~~
~~à moitié effacées est encore parfaitement lisible.~~

Si on liasse la perimère et la drnièree ltretes à la bnnoe pclae dnas cquahe mot, arols ctete sncedone prshae est asusi pfatitaerment liiblse (ou psqeure!)

- Pourquoi tant de redondance ?
  - pour mieux se comprendre, tout simplement !
  - pour être capable de lire un texte contenant des fotes d'ortografe.

Exemples de compression dans la vie de tous les jours

```
print("bla bla bla bla bla ")
print(5*"bla ")
```

- **Langage SMS** : “slt”, “tqt”, “mdr”, “jpp” ...
- **Acronymes** : EPFL, UNIL, ...
- **Code Morse** : A = “.-”, E = “.”, S = “...”, T = “-”,
  tandis que X = “-..-”, Z = “--..”

Exemple de compression (sans pertes)

```
print("bla bla bla bla bla ")
print(5*"bla ")
```

- **MONTREUX**

- **LAUSANNE**

Exemple de compression (sans pertes)

```
print("bla bla bla bla bla ")
print(5*"bla ")
```

Essayons d'encoder les séquences de lettres **MONTREUX** et **LAUSANNE** sous la forme de séquences de 0 et de 1 :

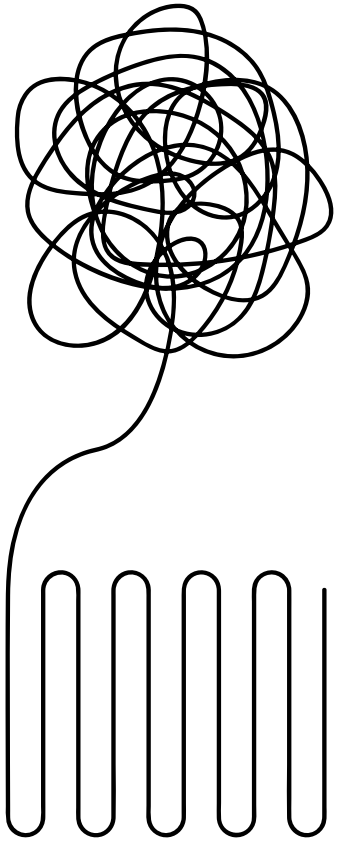
- La séquence **MONTREUX** a huit lettres différentes : il n'y a donc pas de choix pour cette séquence : 3 bits par lettre sont nécessaires (par exemple) :
  M ↔ 000, **O** ↔ 001, etc. et donc $3\ x\ 8 = 24$ bits en tout.

- Dans la séquence **LAUSANNE** par contre, les lettres **A** et **N** se répètent chacune deux fois ; on peut donc abréger la représentation de celles-ci en utilisant le code suivant (par exemple) :
  A ↔ 11, **N** ↔ 10, **L** ↔ 000, **S** ↔ 001, **U** ↔ 010, **E** ↔ 011.

  Cette représentation n'utilise que $2\ x\ 4\ + 3\ x\ 4 = 20$ bits en tout.

Aujourd'hui

- Compression des données
- Entropie
- Algorithmes de Shannon-Fano et Huffman
- Théorème de Shannon
- Compression avec pertes



Exemple 1

Voici une séquence de 16 lettres :

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

Jeu n° 1 :
Deviner quelle lettre a été tirée au hasard en posant un nombre minimum de questions binaires, auxquelles on ne répond que par oui ou par non.

Solution :

- 4 questions sont nécessaires (algorithme de dichotomie).
- On dit que l'entropie de cette séquence est égale à 4.
- Remarquez que $16 = 2^4$, autrement dit : $4 = \log_2(16)$

Exemple 2

Voici une autre séquence de 16 lettres (sans compter les espaces):

IL FAIT BEAU A IBIZA

Jeu n° 2 : Le jeu est le même qu'avant !

Remarques:

- La position de la lettre est tirée au hasard, de manière uniforme.
- Il ne faut deviner que la lettre elle-même (I, L, F, A, ...), pas sa position.

Combien de questions binaires *en moyenne*
sont-elles nécessaires pour deviner la lettre ?

Exemple 2

IL FAIT BEAU A IBIZA

Solution: classer les lettres dans l'ordre décroissant du nombre d'apparitions dans la séquence :

Lettre	I	A	B	L	F	T	E	U	Z
Nombre d'apparitions	4	4	2	1	1	1	1	1	1

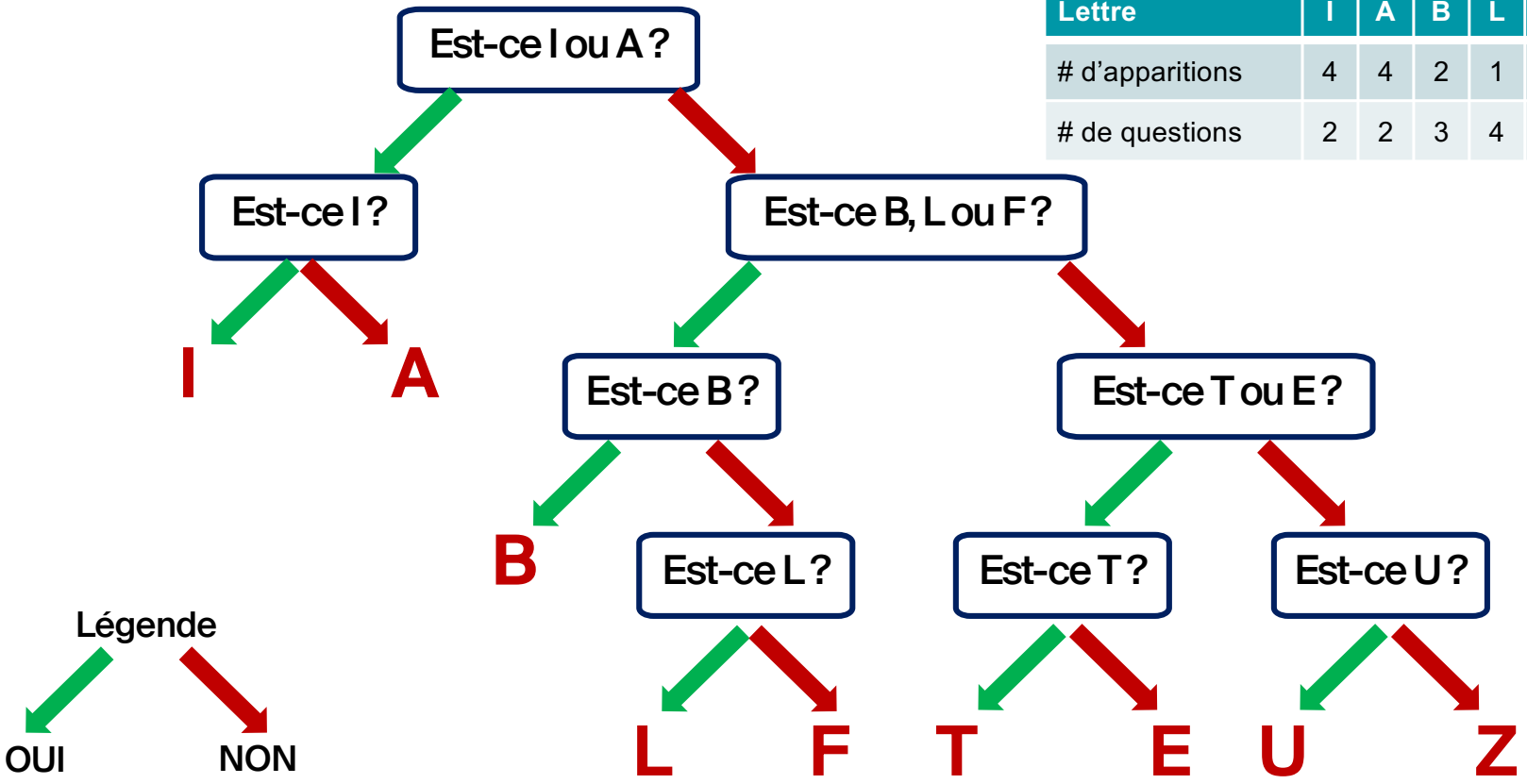
Idée (dichotomie plus générale) : séparer l'ensemble des lettres en deux parties égales en tenant compte de leur nombre d'apparitions, ce qui donne :

- **Question n° 1 :** Est-ce que la lettre est **un I ou un A** ?
  - Si la réponse est **oui** : **Question n° 2:** Est-ce que la lettre est **un I** ?
  - Si la réponse est **non** : **Question n° 2:** Est-ce que la lettre est **un B, L ou F** ?

  etc.

Exemple 2 : Arbre des questions

Lettre	I	A	B	L	F	T	E	U	Z
# d'apparitions	4	4	2	1	1	1	1	1	1
# de questions	2	2	3	4	4	4	4	4	4



Exemple 2 : Solution

Lettre	I	A	B	L	F	T	E	U	Z
Probabilité	$^4/_{16}$	$^4/_{16}$	$^2/_{16}$	$^1/_{16}$	$^1/_{16}$	$^1/_{16}$	$^1/_{16}$	$^1/_{16}$	$^1/_{16}$
Nombre de questions	2	2	3	4	4	4	4	4	4

Nombre de questions à poser en moyenne :

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

On dit que **l'entropie** de cette séquence est égale à 2.875.

Exemple 2

Lettre	I	A	B	L	F	T	E	U	Z
Probabilité p	$4/16$	$4/16$	$2/16$	$1/16$	$1/16$	$1/16$	$1/16$	$1/16$	$1/16$
Nombre de questions = $\log_2(1/p)$	2	2	3	4	4	4	4	4	4

Remarques :

- Pour deviner une lettre qui apparaît 1 fois sur 16, on a besoin de 4 questions. $4 = \log_2(16)$
- Pour deviner une lettre qui apparaît 2 fois sur 16 (i.e. 1 fois sur 8), on a besoin de 3 questions. $3 = \log_2(8)$
- Pour deviner une lettre qui apparaît 4 fois sur 16 (i.e. 1 fois sur 4), on a besoin de 2 questions. $2 = \log_2(4)$

En résumé : **pour deviner une lettre qui apparaît avec une probabilité p, on a besoin de $\log_2\left(\frac{1}{p}\right)$ questions.**

Entropie : Définition générale

- Soit X une séquence de lettres provenant d'un alphabet $A = \{a_1, \ldots, a_n\}$.
- Soit p_j la probabilité d'apparition de la lettre a_j dans la séquence X (remarquez que $0 \leq p_j \leq 1 \; \forall j$ et que $p_1 + \cdots + p_n = 1$).

L'entropie de la séquence X est définie par :

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

Remarque : par convention, si $p_j = 0$, alors on pose $p_j \cdot \log_2\left(\frac{1}{p_j}\right) = 0$.

Exemple 3

Voici encore une autre séquence de 16 lettres :

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

Jeu n° 3: Le jeu est encore le même qu'avant !

- Cette fois-ci, aucune question n'est nécessaire pour deviner la lettre choisie!
- On dit que l'entropie de cette séquence est égale à 0.

Entropie : Exemple

Lettre	I	A	B	L	F	T	E	U	Z
Probabilité p	$^4/_{16}$	$^4/_{16}$	$^2/_{16}$	$^1/_{16}$	$^1/_{16}$	$^1/_{16}$	$^1/_{16}$	$^1/_{16}$	$^1/_{16}$
Nombre de questions = $\log_2(^1/_p)$	2	2	3	4	4	4	4	4	4

$$H(X) = 2 \cdot \frac{1}{4} \cdot \log_2(4) + 1 \cdot \frac{1}{8} \cdot \log_2(8) + 6 \cdot \frac{1}{16} \cdot \log_2(16) = 2.875$$

Notez qu'en général, l'entropie ne coïncide pas avec le nombre moyen de questions binaires à poser pour deviner une lettre !

Exemple : Trouver une lettre tirée au hasard dans la séquence **ABB** nécessite exactement 1 question binaire, mais

$$H(X) = \frac{1}{3} \cdot \log_2(3) + \frac{2}{3} \cdot \log_2\left(\frac{3}{2}\right) \simeq 0.92$$

Origines de la notion d'entropie

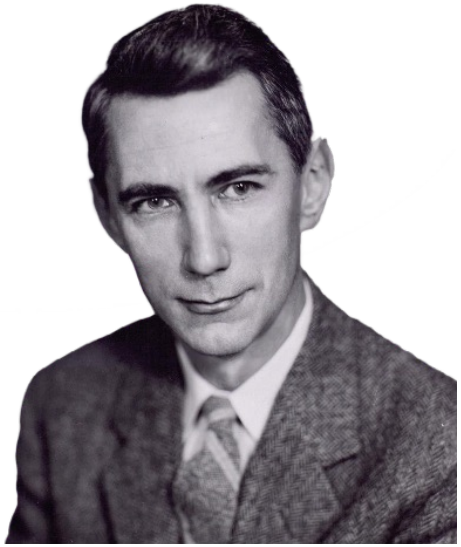
Origine en physique (Boltzmann, 1872) :

L'entropie mesure le "désordre" dans un système physique.

- Ludwig Boltzmann (1844-1905)
- ardent défenseur de l'existence des atomes
- père de la physique statistique

Théorie de l'information (Shannon, 1948) :

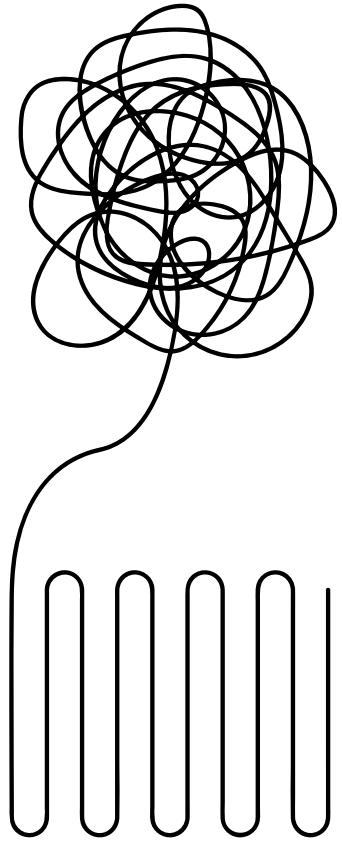
L'entropie mesure la "quantité d'information" contenue dans un signal.



- Claude Shannon (1916-2001)
- mathématicien, ingénieur électricien, cryptologue,
- père de la théorie de l'information, jongleur...

EPFL

Interprétation de la notion d'entropie



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

IL FAIT BEAU A IBIZA $H(X) = 2.875$

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

- Plus il y a de **lettres différentes**, plus il y a de désordre, plus il y a de **nouveauté** et donc "**plus d'information**" dans le message.
- Plus il y a de **lettres semblables**, moins il y a de désordre, plus il y a de **redondance** et donc "**moins d'information**" dans le message.

Quelques propriétés de l'entropie

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

- Pour une probabilité d'apparition $0 \leq p \leq 1$ donnée, $\log_2\left(\frac{1}{p}\right) \geq 0$.
- $H(X) \geq 0$ en général, et $H(X) = 0$ si et seulement si l'ordre est total (c'est-à-dire si toutes les lettres sont les mêmes).
- Si n est la taille de l'alphabet utilisé, $H(X) \leq \log_2(n)$ en général et $H(X) = \log_2(n)$ si et seulement si le désordre est total (c'est-à-dire si toutes les lettres sont différentes).

Entropie : quelques rappels

L'entropie d'une séquence X est définie par :

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

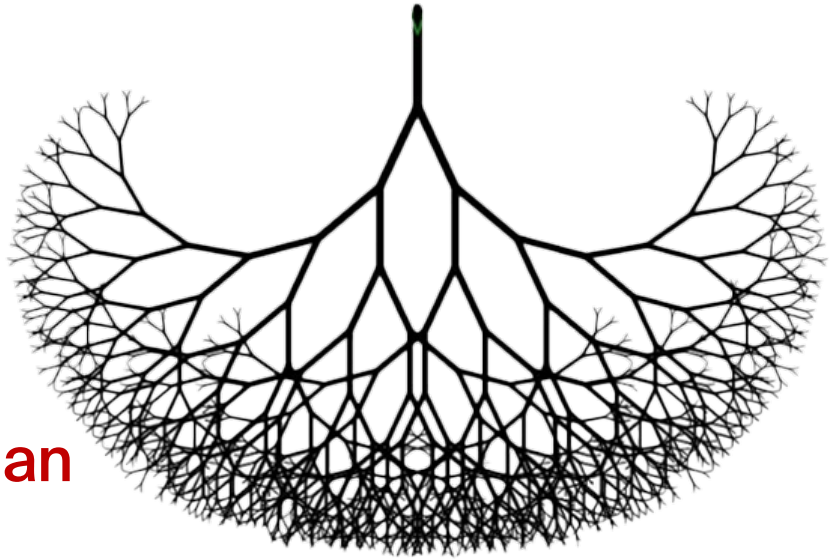
où $p_1, \ldots, p_n$ sont les probabilités d'apparition des lettres $a_1, \ldots, a_n$ dans la séquence X.

$H(X)$ ne dépend donc pas de l'ordre des lettres, ni de leurs valeurs!

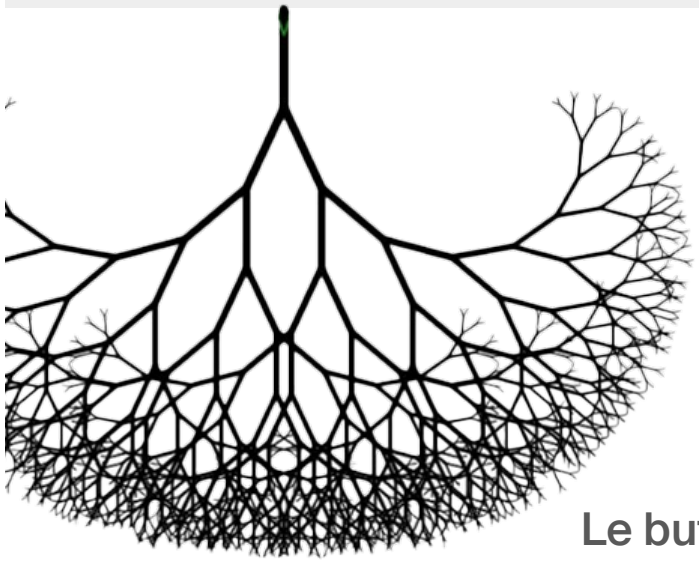
$$H(BANANA) = H(ANANAS) = H(BONOBO)$$

Aujourd'hui

- Compression des données
- Entropie
- Algorithmes de Shannon-Fano et Huffman
- Théorème de Shannon
- Compression avec pertes



Compression sans pertes



Principe : Utiliser la redondance présente dans les données (en pratique, abréger ce qui revient souvent)

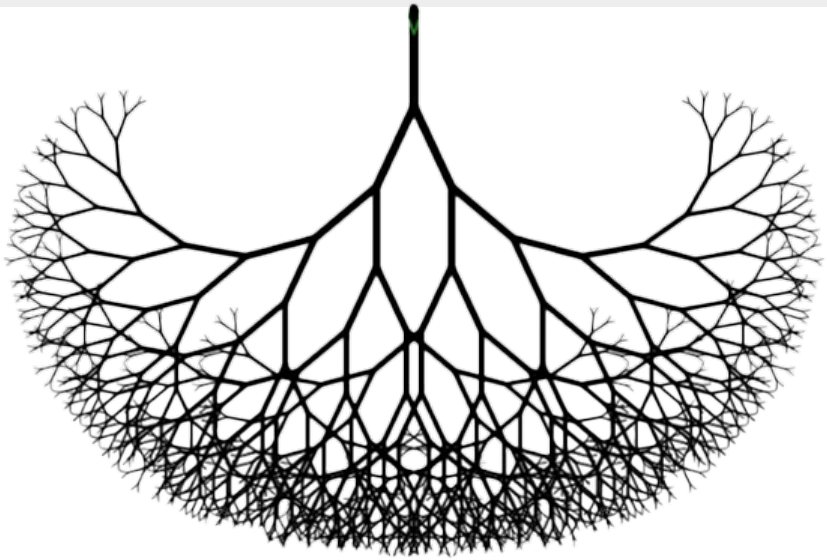
Exemple : Voici une séquence de 13 lettres (en incluant les deux points d'exclamation):

A B R A C A D A B R A !!

Le but est d'encoder cette séquence de lettres en une séquence de bits, en utilisant le moins de bits possibles et en respectant les règles suivantes :

- A chaque lettre correspond un unique mot de code (ex : A ↔ 01).
  L'ensemble des mots de code constitue un dictionnaire.
- La séquence de bits ainsi produite doit être décodable de manière unique à l'aide du dictionnaire.

Compression sans pertes



Deux approches :

1. En suivant une version légèrement modifiée du jeu des questions, on obtient **l'algorithme de Shannon-Fano**.

2. En regroupant les lettres au fur et à mesure selon leurs probabilités d'apparition, on obtient **l'algorithme de Huffman**.

Optimal

Algorithme de Shannon-Fano

Vous avez déjà rencontré Claude Shannon...

Voici Robert Fano (1917-2016), prof. au MIT

Reprenons le jeu des questions sur notre nouvel exemple :

A B R A C A D A B R A !!

- Comme précédemment, on commence par classer les lettres de la séquence dans **l'ordre décroissant** de leur nombre d'apparitions :

Lettre	A	B	R	!	C	D
Nombre d'apparitions	5	2	2	2	1	1

- Quelle est la meilleure première question à poser ici ?

Algorithme de Shannon-Fano

A B R A C A D A B R A !!

Lettre	A	B	R	!	C	D
Nombre d'apparitions	5	2	2	2	1	1

- **Problème :** On ne peut pas diviser l'ensemble des lettres en deux parties strictement égales (tout en respectant l'ordre décroissant).

- **Solution :** On minimise la différence de taille entre les deux ensembles

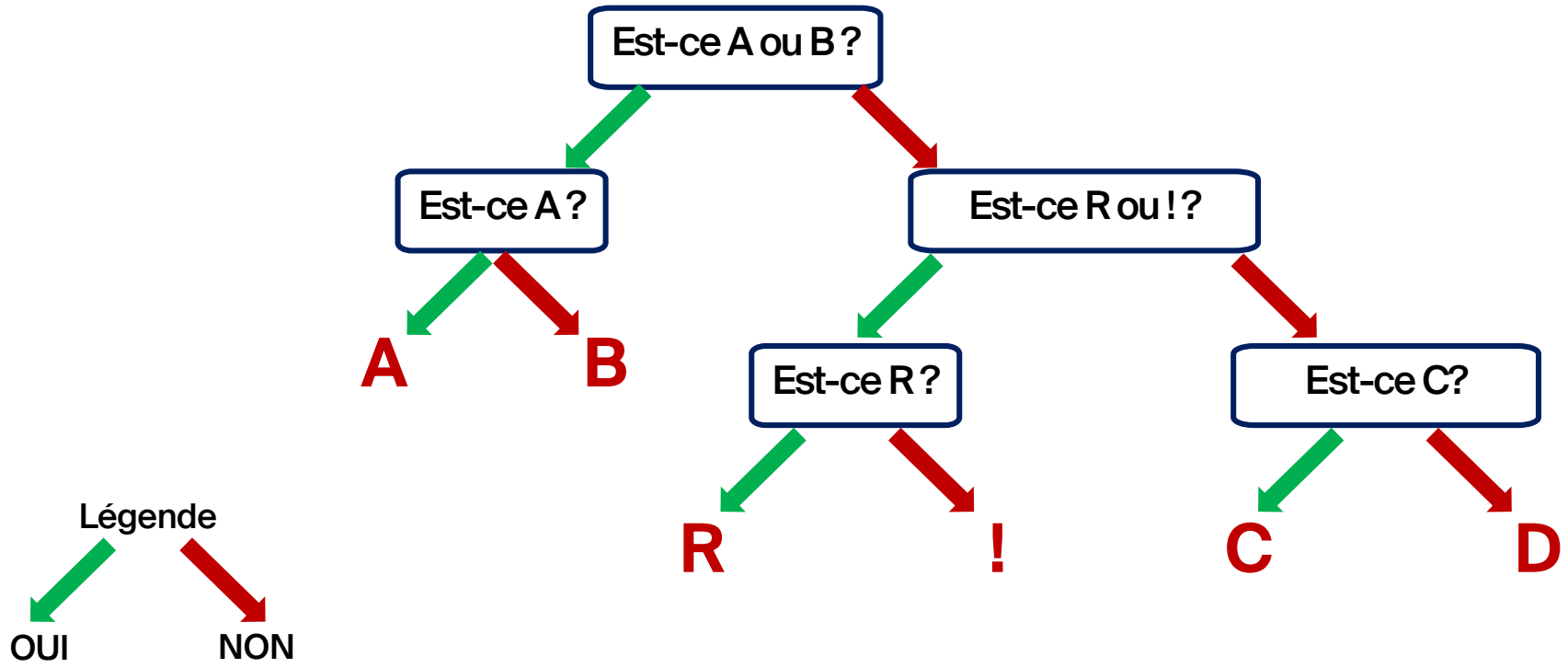
→ **Q1:** "Est-ce un A ou un B ?" (→ 7 et 6 possibilités, respectivement)

- **Si la réponse est oui,** quelle est la 2ème question à poser ?
  → **Q2:** « Est-ce un A ? » (on n'a pas tellement le choix ici...)

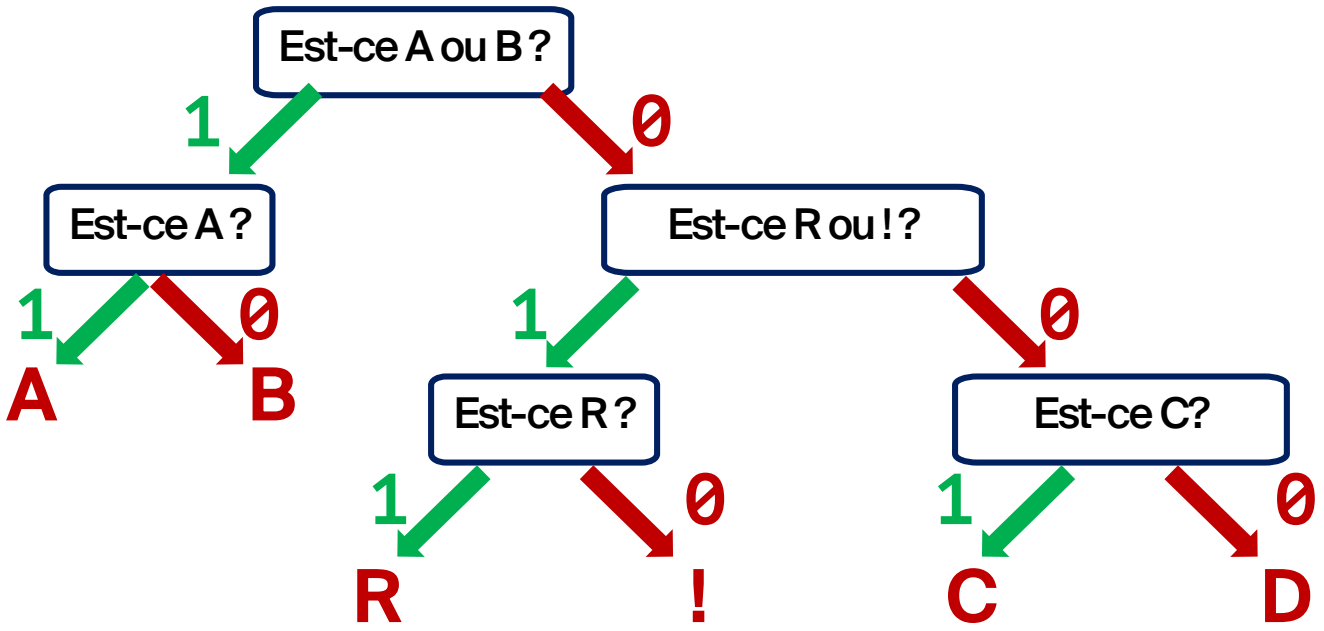
- **Si la réponse est non,** quelle est la 2ème question à poser ?
  → **Q2:** « Est-ce un R ou un "!" ? » ou **Q2:** « Est-ce un R ? »
  On a le choix ici ! (mais aucun indice sur quel est le meilleur...)

Algorithme de Shannon-Fano: Option 1

Lettre	A	B	R	!	C	D
# d'apparitions	5	2	2	2	1	1



Algorithme de Shannon-Fano: Option 1



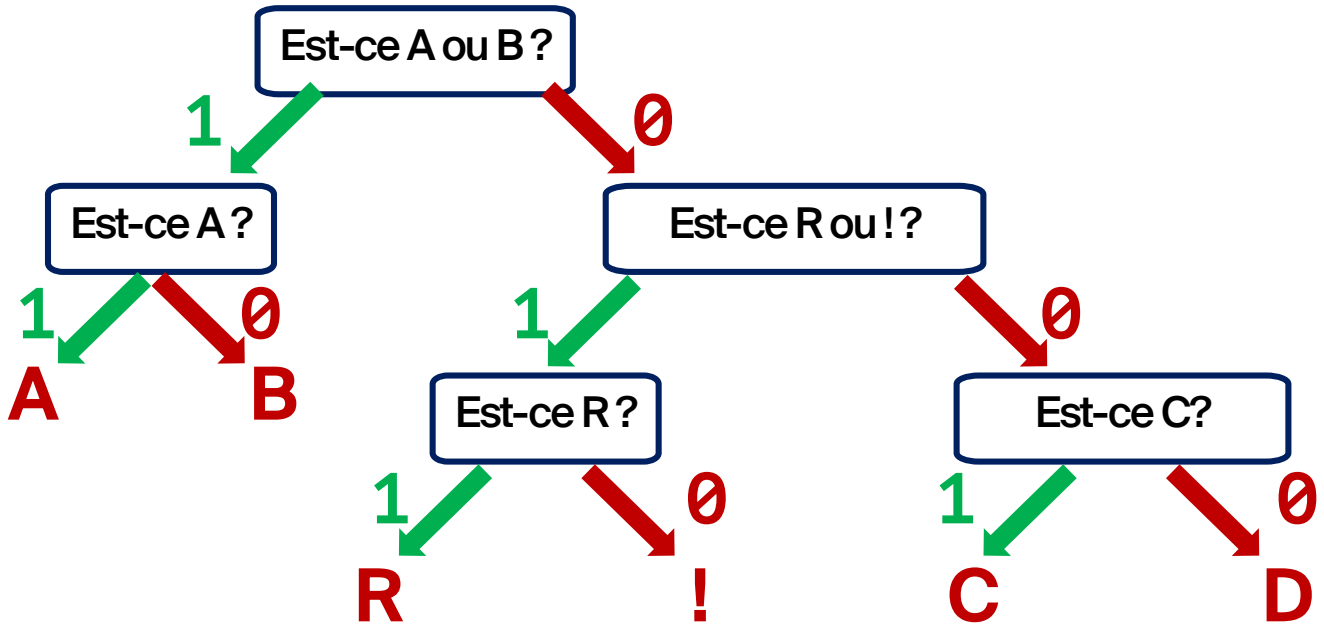
Construction du dictionnaire :

- **Règle n° 1 :** Le nombre de bits attribués à chaque lettre est égal au nombre de questions nécessaires pour la deviner.
- **Règle n° 2 :** Les bits 1 ou 0 sont attribués en fonction des réponses « oui » ou « non » obtenues aux questions.

Algorithme de Shannon-Fano: Option 1

A B R A C A D A B R A !!

Lettre	A	B	R	!	C	D
Nombre d'apparitions	5	2	2	2	1	1
Nombre de questions	2	2	3	3	3	3
Mot de code	11	10	011	010	001	000



Algorithme de Shannon-Fano: Option 1

- Dictionnaire résultant :

Lettre	A	B	R	!	C	D
Nombre d'apparitions	5	2	2	2	1	1
Nombre de questions	2	2	3	3	3	3
Mot de code	11	10	011	010	001	000

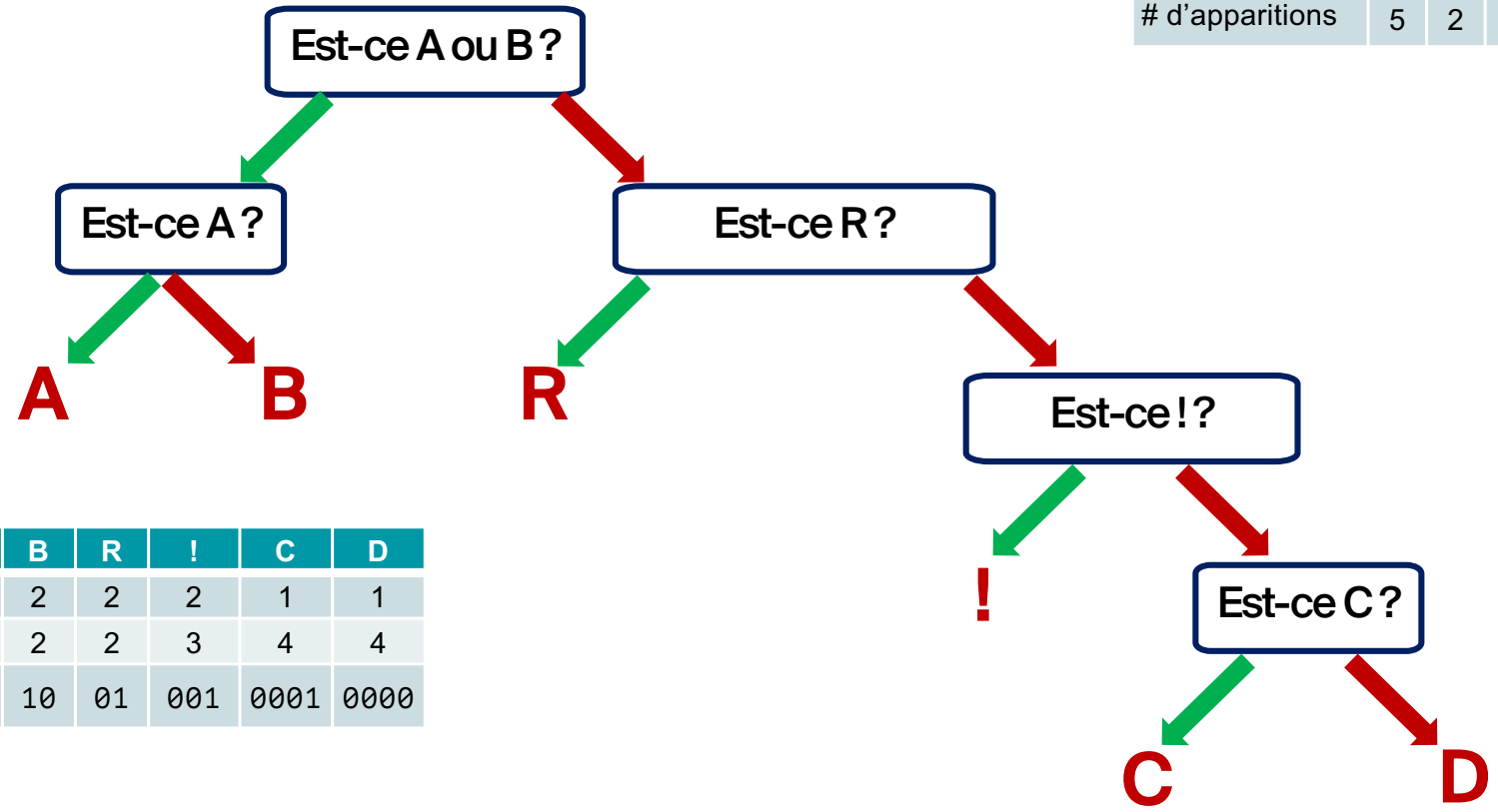
- Pour encoder la séquence **A B R A C A D A B R A !!**, on utilise donc $2 \cdot 7 + 3 \cdot 6 = 32$ bits au total, ce qui représente $\frac{32}{13} \approx 2.46$ bits par lettre.
- Séquence de bits produite :

```
11100111100111000111001111010010
```

- Cette séquence est parfaitement décodable à l'aide du dictionnaire, si on la lit de gauche à droite car aucun mot de code du dictionnaire n'est le préfixe d'un autre.

Algorithme de Shannon-Fano: Option 2

Lettre	A	B	R	!	C	D
# d'apparitions	5	2	2	2	1	1



Lettre	A	B	R	!	C	D
# d'apparitions	5	2	2	2	1	1
# de questions	2	2	2	3	4	4
Mot de code	11	10	01	001	0001	0000

Algorithme de Shannon-Fano: Option 2

- Dictionnaire résultant :

Lettre	A	B	R	!	C	D
Nombre d'apparitions	5	2	2	2	1	1
Nombre de questions	2	2	2	3	4	4
Mot de code	11	10	01	001	0001	0000

- Nombre de bits utilisés pour encoder **A B R A C A D A B R A !!** :

$2 \cdot 9 + 3 \cdot 2 + 4 \cdot 2 = 32$ bits au total (donc $\frac{32}{13} \approx 2.46$ bits par lettre)

Remarque : Ici, le nombre total de bits utilisés est le même pour les deux options choisies, mais ce n'est pas le cas en général pour l'algorithme de Shannon-Fano.

Algorithme de Huffman

- David Albert Huffman (1925-1999)
  - ingénieur électricien, pionnier de l'informatique
  - élève de Robert Fano...

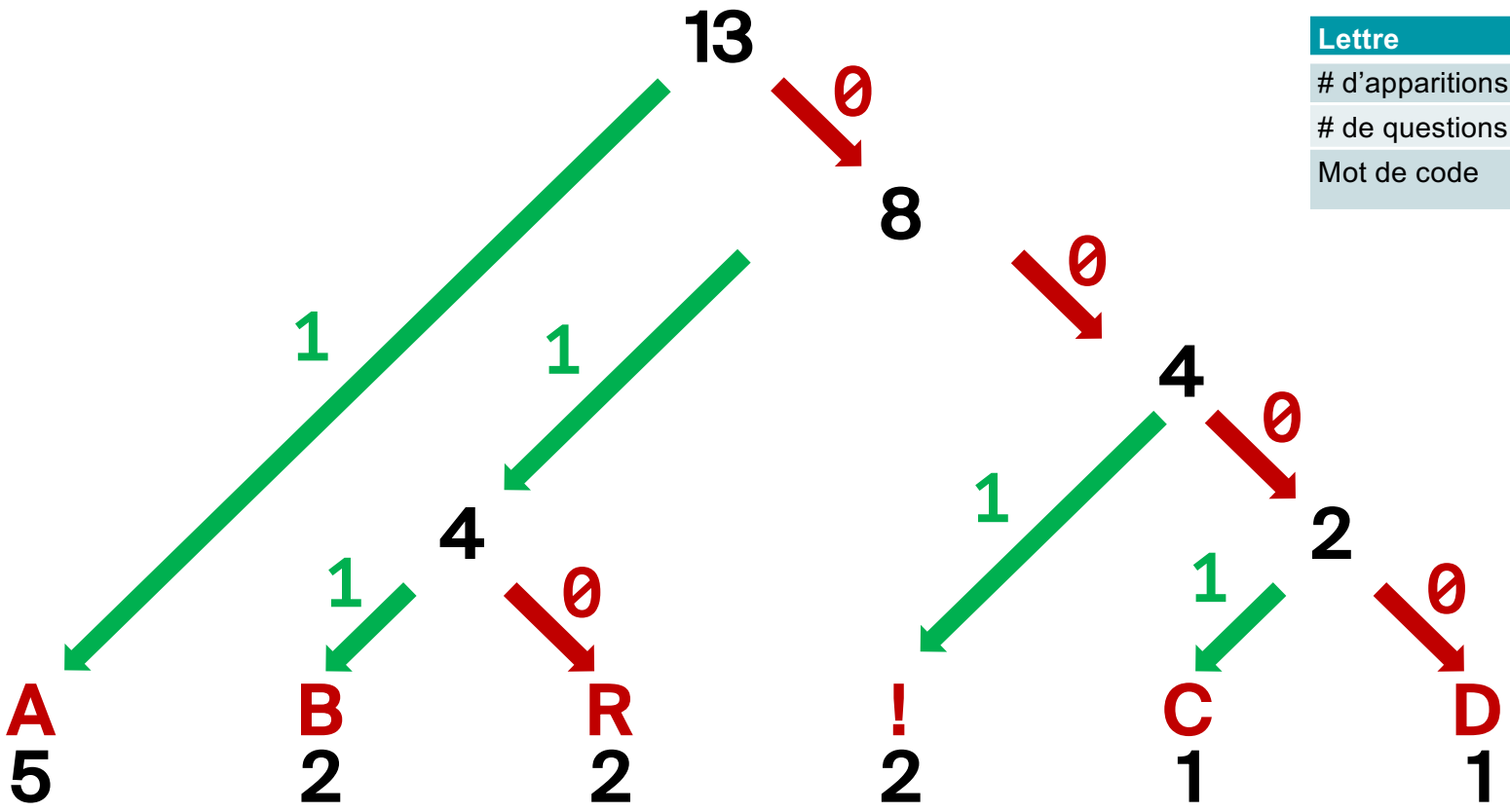
Repartons avec le tableau des nombres d'apparitions de chaque lettre :

Lettre	A	B	R	!	C	D
Nombre d'apparitions	5	2	2	2	1	1

Idée principale : Au lieu de construire l'arbre depuis le haut, construisons-le **depuis le bas** !

- Voici l'algorithme :
  1. Regrouper les deux lettres les moins probables (ici, C et D)
  2. Considérer ces deux lettres comme une nouvelle "lettre" commune, et faire la somme des nombres d'apparitions (ici, $1 + 1 = 2$)
  3. Recommencer en 1. jusqu'à ce qu'il ne reste qu'une seule "lettre"

Algorithme de Huffman



Lettre	A	B	R	!	C	D
# d'apparitions	5	2	2	2	1	1
# de questions	1	3	3	3	4	4
Mot de code	1	011	010	001	0001	0000

Algorithme de Huffman

- Dictionnaire résultant:

Lettre	A	B	R	!	C	D
Nombre d'apparitions	5	2	2	2	1	1
Nombre de questions	1	3	3	3	4	4
Mot de code	1	011	010	001	0001	0000

- Nombre de bits par lettre pour encoder la séquence **ABRACADABRA!!** :

$$1 \cdot 5 + 3 \cdot 6 + 4 \cdot 2 = 31, \text{ donc } \frac{31}{13} \approx 2.38$$

→ **mieux** que Shannon-Fano !

De plus :

- L'exécution de l'algorithme de Huffman est aussi plus rapide!
- En général, il y a aussi des choix à faire avec cet algorithme, mais le nombre total de bits utilisés est toujours le même.

Et l'entropie, dans tout ça ?

Lettre	A	B	R	!	C	D
Probabilités	$^5/_{13}$	$^2/_{13}$	$^2/_{13}$	$^2/_{13}$	$^1/_{13}$	$^1/_{13}$

- L'entropie de la séquence **A B R A C A D A B R A !!** vaut :

$$H(X) = \frac{5}{13} \cdot \log_2\left(\frac{13}{5}\right) + 3 \cdot \frac{2}{13} \cdot \log_2\left(\frac{13}{2}\right) + 2 \cdot \frac{1}{13} \cdot \log_2\left(\frac{13}{1}\right) \approx 2.35$$

ce qui est légèrement plus petit que le nombre moyen de bits par lettre utilisés par chacun des deux algorithmes (respectivement 2.46 et 2.38).

Nous allons voir que c'est toujours le cas!

Aujourd'hui

- Compression des données
- Entropie
- Algorithmes de Shannon-Fano et Huffman
- Théorème de Shannon
- Compression avec pertes

Définitions

- Un code binaire C est dit **sans préfixe** si aucun mot de code n'est le préfixe d'un autre.

Lettre	A	B	R	!	C	D
# d'apparitions	5	2	2	2	1	1
Mot de code	1	011	010	001	0001	0000

- **longueur moyenne du code** $L(C)$

Définitions

- Un code binaire est un ensemble C dont les éléments $c_1, \dots, c_n$ (également appelés mots de code) sont des suites de `0` et de `1` de longueur finie.

Exemple : $C = \{\,$`11, 10, 01, 001, 0001, 0000`$\,\}$

- On note l_j la longueur d'un mot de code c_j.

Exemple : $l_4 = 3$

- Un code binaire C est dit **sans préfixe** si aucun mot de code n'est le préfixe d'un autre. Ceci garantit :
  - que tous les mots de code sont différents ;
  - que tout message formé de ces mots de code peut être décodé au fur et à mesure de la lecture (si celle-ci se fait de gauche à droite).

Exemple : Avec le code binaire ci-dessus, la séquence `111001110001111` se lit `11, 10, 01, 11, 0001, 11`

Définitions (suite)

- Le code binaire $C = \{c_1, \dots, c_n\}$ peut être utilisé comme un dictionnaire pour représenter une séquence X formée avec des lettres tirées d'un alphabet $\mathcal{A} = \{a_1, \dots, a_n\}$. Chaque lettre a_j est représentée par le mot de code c_j de longueur l_j.

- Si les lettres $a_1, \dots, a_n$ apparaissent avec des probabilités $p_1, \dots, p_n$ dans la séquence X, alors la **longueur moyenne du code** (i.e., le nombre moyen de bits utilisés par lettre) est donnée par

$$L(C) = p_1 \cdot l_1 + \cdots + p_n \cdot l_n$$

Théorème de Shannon

Quel que soit le code binaire C **sans préfixe** utilisé pour représenter une séquence X donnée, l'inégalité suivante est toujours vérifiée :

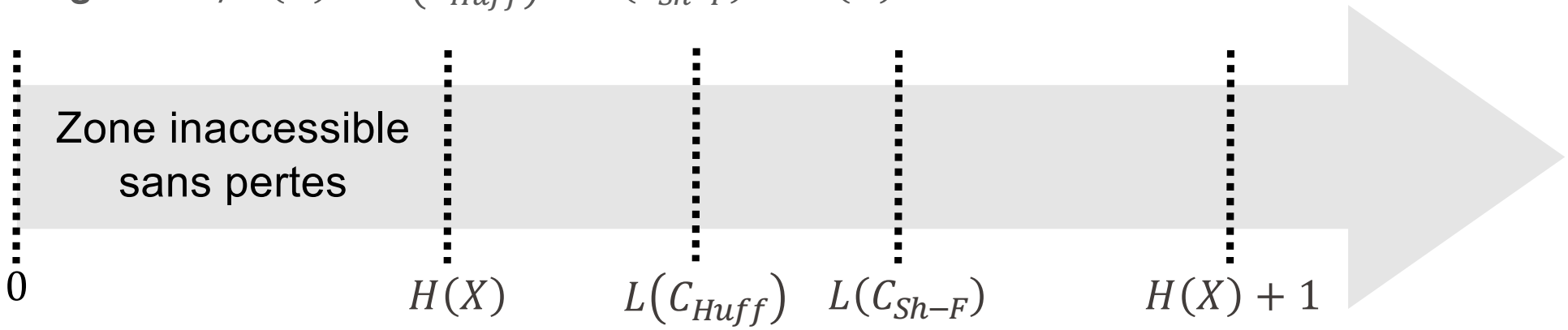
$$H(X) \leq L(C)$$

On voit apparaître ici **un seuil** (cf. théorème d'échantillonnage) :
en-dessous de ce seuil, il n'est pas possible de compresser des données sans faire de pertes.

Quelques informations supplémentaires

On peut montrer de plus les inégalités suivantes :

- En général, $H(X) \leq L\left(C_{Huff}\right) \leq L(C_{Sh-F}) < H(X) + 1$:



- Pour **n'importe quel code binaire sans préfixe** C, on a de plus

$$L\left(C_{Huff}\right) \leq L(C)$$

ce qui veut dire que **le code de Huffman est optimal** !

Aujourd'hui

- Compression des données
- Entropie
- Algorithmes de Shannon-Fano et Huffman
- Théorème de Shannon
- **Compression avec pertes**

Compression avec pertes

- Pour compresser des données sans pertes, on ne peut donc pas descendre en-dessous de la borne de Shannon.
- Si on essaye malgré tout en utilisant un algorithme du même type, on court à la catastrophe !
- Retournons à notre exemple **A B R A C A D A B R A !!** et essayons d'utiliser code binaire suivant (**avec** préfixe) :

Lettre	A	B	R	!	C	D
Mot de code	1	0	11	10	01	00

- Avec un tel code, on n'utilise que 19 bits au total, et non 31, mais la représentation binaire du message donne : `10111`...
- En lisant de gauche à droite, on ne peut pas décoder le début de la séquence: est-ce **A B A A A** ? **A B R A** ? **A C R** ? ou encore **! R A** ? ...

Compression avec pertes

Cependant, on est parfois **obligé** de compresser en faisant des pertes :

1. $\mathbb{R}$
2. $f_{max} = \infty$
3. lorsqu'on désire télécharger sur un site web l'intégralité de ses photos de vacances (quelques gigaoctets pour une centaine de photos avec la plus haute résolution).

Comment procéder dans les deux derniers cas ?

- Même principe : appliquer un **filtre passe-bas** pour éliminer les hautes fréquences !

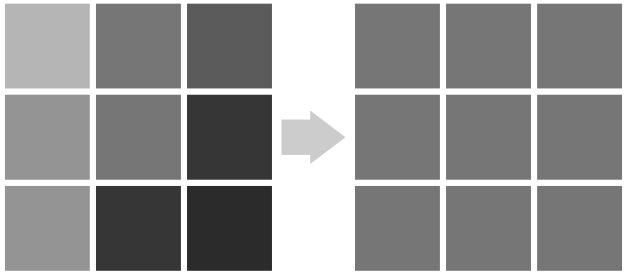
Compression avec pertes : Images

- L'oreille humaine ne perçoit pas des sons au-delà de $\sim 20\ kHz$...
- De même, l'œil humain a un pouvoir de résolution d'environ une minute d'arc = $\frac{1}{60} \sim 0.017$ degré, ce qui veut dire qu'il ne distingue pas :
  - des cratères sur la lune d'un diamètre inferieur à 100km ;
  - des objets de taille inférieure à 1 mm situés à 3 m de distance
  - des pixels de taille inférieure à 0.2×0.2mm sur un écran d'ordinateur (à 50 cm de distance).

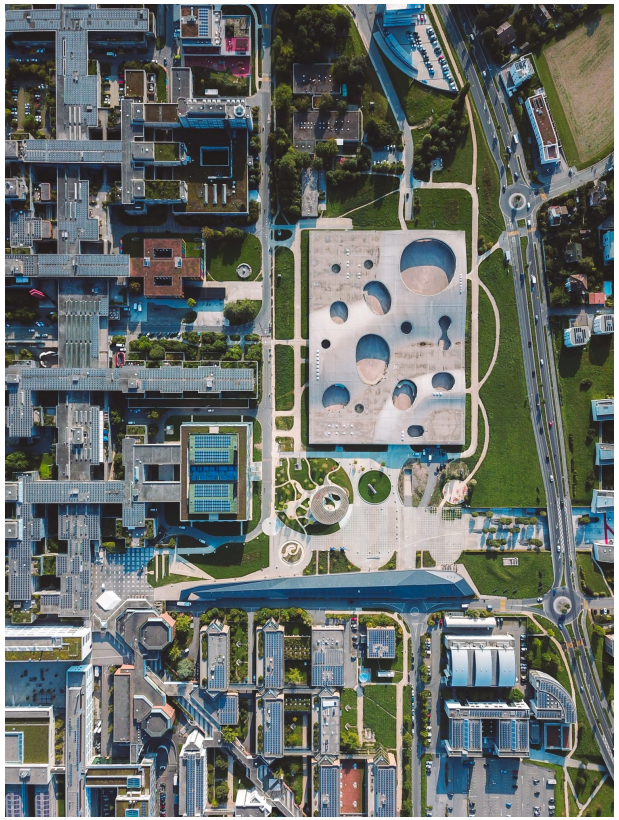
Comment filtrer les hautes fréquences (spatiales) dans une image ?

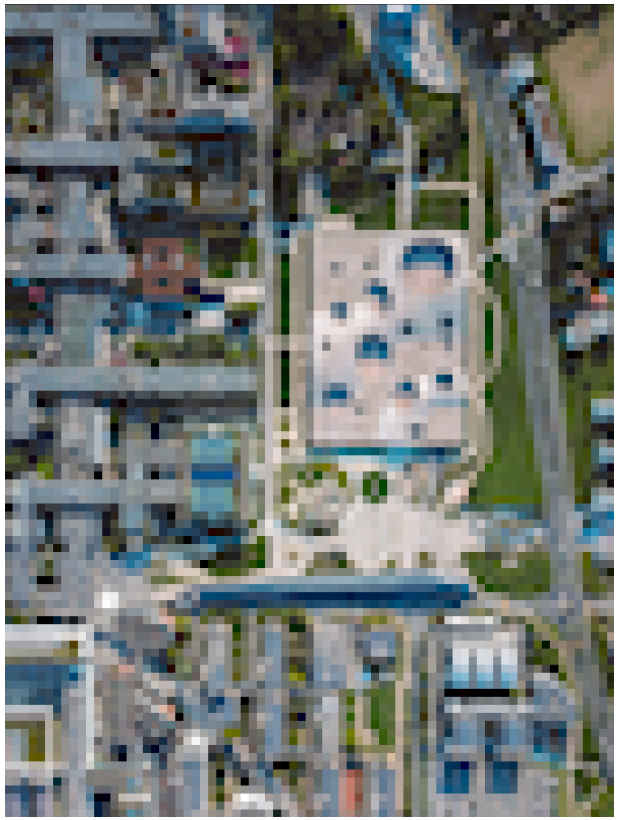
Une façon simple de faire : moyenner les couleurs sur des zones de plus ou moins grande taille.

C'est un filtre à moyenne mobile !



Compression avec pertes : Images





Compression avec pertes : Images

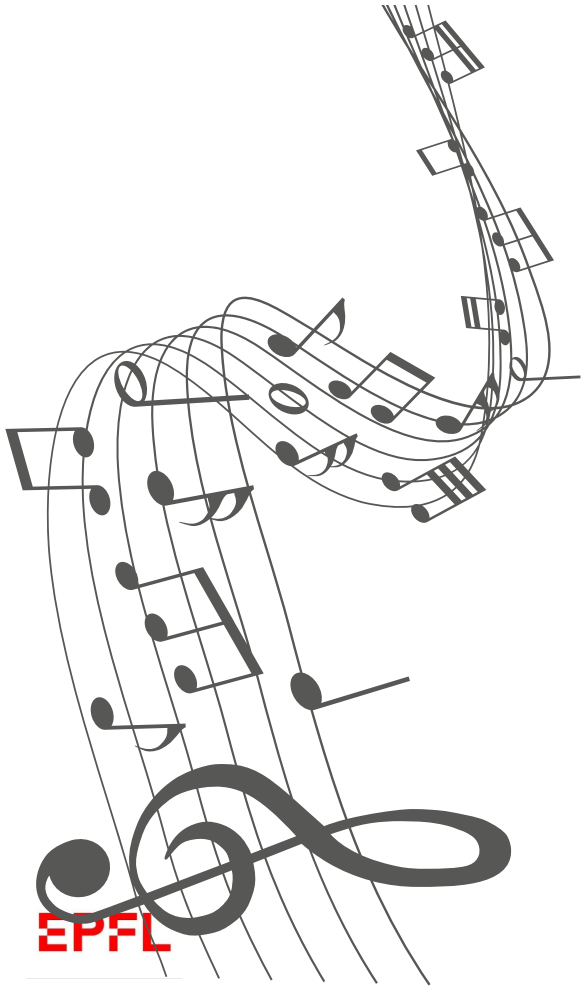
On voit apparaître un **compromis** :

- plus on utilise des pixels de grande taille, moins on a besoin d'**espace-mémoire** pour stocker l'image...
- mais plus l'image d'origine est déformée : on parle de **distorsion**.

Il existe bien sûr des algorithmes beaucoup plus sophistiqués pour compresser une image avec pertes :

- format JPEG :
  - analyse les fréquences spatiales présentes dans l'image
  - n'en retient que les plus basses
  - utilise un algorithme de compression sans pertes par-dessus le tout
- format JPEG 2000 : la même chose, mais avec des **ondelettes**.

Compression avec pertes : Son

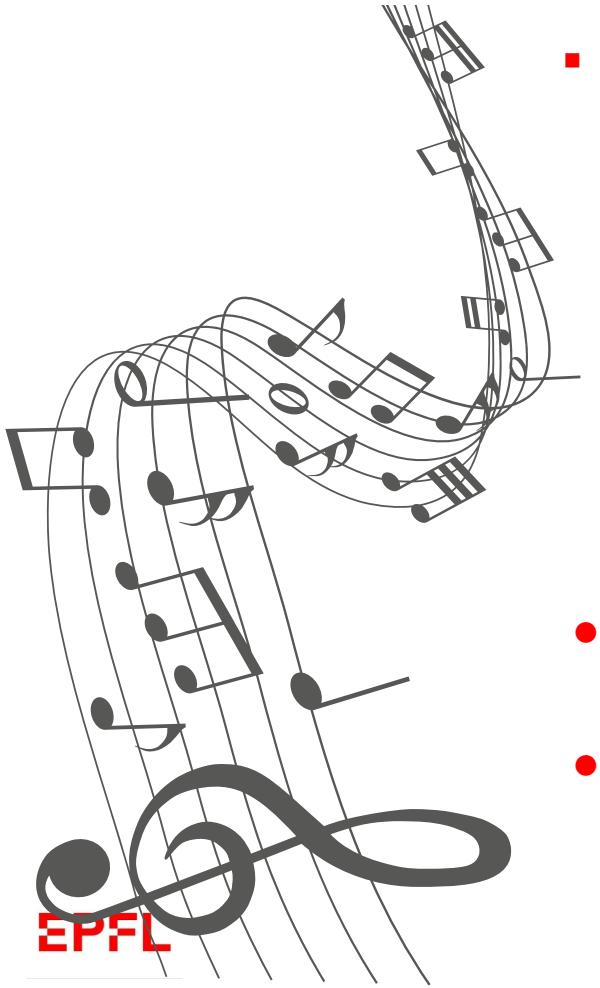


Et pour le son, comment procéder ?

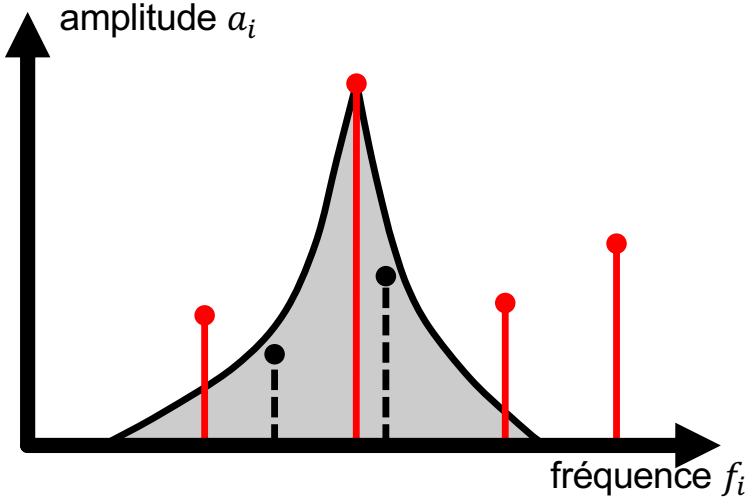
- Pour rappel, le son enregistré sur un CD est d'abord filtré à $22\ kHz$, puis échantillonné à $44.1\ kHz$, et chaque échantillon est encodé sur $2 \cdot 16$ bits (2 canaux pour la stéréo).
- Pour enregistrer une seconde, il faut donc $44'100 \cdot 16 \cdot 2 \approx 1.4$ mégabits.
- Le format MP3 permet d'encoder cette information sur 128 kilobits seulement, ce qui correspond à une réduction d'environ 90% de la taille d'un fichier ! (sans déformation **sensiblement audible** du son)

Comment est-ce possible ?

Compression avec pertes : Son



- C'est (en partie) grâce à **l'effet de masque** : lorsqu'une sinusoïde avec une certaine fréquence est présente avec grande amplitude dans un son, elle cache à l'oreille humaine les autres sinusoïdes de fréquences proches et de moindre amplitude (c'est un effet **psychoacoustique**).



- En conséquence, il n'y a pas besoin d'encoder une partie du signal, car on ne l'entend de toute façon pas !
- En se basant sur ce principe, Karlheinz Brandenburg et al. ont créé le format **MP3** en 1993.

Aujourd'hui

- Compression des données
- Entropie
- Algorithmes de Shannon-Fano et Huffman
- Théorème de Shannon
- Compression avec pertes

Résumé Semaine 6 – ICC-T

- La compression sans pertes permet de réduire la taille d'un fichier sans perdre d'information, en exploitant la redondance.
- L'entropie mesure l'information moyenne par lettre dans une source :

$$H(X) = \sum p_j \log_2 \left(\frac{1}{p_j}\right)$$

- Les algorithmes de Shannon-Fano et Huffman construisent des codes binaires sans préfixe qui attribuent des mots de code plus courts aux lettres fréquentes.
- La longueur moyenne du code (par lettre) est donnée par :

$$L(C) = \sum p_j \cdot l_j$$

- Théorème de Shannon :

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

  - L'entropie est une borne théorique minimale : on ne peut pas aller en-dessous sans pertes.
- La compression avec pertes permet de réduire davantage la taille (images, son, vidéo), au prix d'une distorsion contrôlée (ex. : JPEG, MP3).

rafael.pires@epfl.ch

Merci