Chapitre III

Complexité des données

1 Ordre asymptotique

Rappelons maintenant la notion de borne asymptotique supérieure.

Soit f et g des fonctions de $\mathbb{N}$ dans $\mathbb{R}^+$.

Définition III.1 (grand O)

définition : $f(n) = O(g(n))$ si $\exists c > 0, n_0 \in \mathbb{N}^* \mid \forall n \geq n_0, f(n) \leq c.g(n)$
se lit : g est une borne asymptotique supérieure pour f.
se comprend : f ne grandit pas plus vite que g.

Cette définition s'interprète de la manière suivante :

- On borne supérieurement une fonction $f(n)$ par une fonction $c.g(n)$.
- La constante c ne dépend pas de n. Il n'y a aucune contrainte sur son ordre de grandeur (pourrait être 10^9).
- Le n_0 est le n à partir duquel la relation $f(n) \leq c.g(n)$ est vérifiée.
  On tient compte ainsi de la tendance générale quand n devient assez grand.

Remarques 16:

- attention à l'ordre d'écriture.
- $f(n) = O(g(n)) = O(h(n))$ signifie $f(n) = O(g(n))$ et $g(n) = O(h(n))$
- sert à quantifier la vitesse de croissance d'une fonction le plus souvent croissante.

Exemples 25:

- si $f_1(n) = 3n$ alors $f_1(n) = O(n)$ et $100f_1(n) = O(n)$.
- $f_1(n) = 3n$ et $f_2(n) = 5n$. Et $f_1(n) = O(n)$, $f_2(n) = O(n)$ et $f_1(n) + f_2(n) = O(n)$.
- si $f(n) = 3n^2 + 4n + 5$, on a $3n^2 = O(n^2)$, $4n = O(n)$ et $5 = O(1)$. Alors le terme de plus haut degré domine tous les autres : $O(f(n)) = O(n^2)$.
- $O(3n + 2\log(n)) = O(3n) + O(2\log(n)) = O(n)$: les termes en $\log(n)$ sont dominés par les termes en n.
- $O(4\log(n) + 5) = O(4\log(n)) + O(5) = O(\log(n))$: tout terme constant est dominé par les termes dépendant de n.
- $O(100) = O(1)$

On obtient toujours l'ordre asymptotique à une constante multiplicative près en mettant en facteur le terme dominant et en faisant tendre n vers l'infini. Si on reprend l'exemple ci-dessus : $f(n) = n^2\left(3 + \frac{4}{n} + \frac{5}{n^2}\right)$. Alors $\lim_{n \to \infty} f(n) = 3n^2 = O(n^2)$.

2 Premières approches

2.1 Définition de l'information

Si on considère les chaînes de 24 bits suivantes :

1. 010101010101010101010101
2. 110100110010110100101100
3. 100111011101011100100110

Chacune pourrait représenter 24 tirages d'un pile (0) ou face (1).

Que peut-on dire de ces chaînes ?

- pour la première, c'est 12 répétitions de 01.
- pour la seconde, le $i^{\text{ème}}$ est à 1 si le nombre de 1 dans l'écriture en binaire de i est impair.

i	1	2	3	4	5	6	7	8	9	…
binaire	1	10	11	100	101	110	111	1000	1001	…
résultat	1	1	0	1	0	0	1	1	0	…

- la dernière chaîne n'expose pas de structure visible, et semble aléatoire.

Pourquoi s'intéresser à savoir si une chaîne est aléatoire ou pas ?

- en compression,
  - — une chaîne aléatoire semble ne pas pouvoir être compressée, (= impossibilité de prévoir le symbole suivant ou de trouver une structure).
  - — une chaîne particulière a généralement une représentation plus courte.
- en cryptographie, une chaîne qui contient une information redondante peut être exploitée pour casser le code (cryptographie).
- …

Mais comment savoir si une chaîne est aléatoire ?

2.2 Théorie des probabilités

Si on prend un jeu de pile ou face, la probabilité p de chaque face pour une pièce non truquée est de $\frac{1}{2}$.

Si on considère que les tirages de chaque pile-ou-face sont indépendants, alors toute suite de n tirages est aussi probable que n'importe quelle autre.

Par exemple, la suite de 16 tirages $s_1 = 0000000000000000$ est aussi probable que la suite $s_2 = 1000110111010110$. En effet, dans les deux cas, $\Pr[0] = \Pr[1] = \frac{1}{2}$.

Comme chaque tirage est indépendant, chacune de ces deux chaînes a le même probabilité d'occurrence :

$\Pr[s_1] = \Pr[s_2] = \frac{1}{2^{16}}$.

Avec la théorie classique des probabilités :

- il n'y a **aucun** moyen d'exprimer l'aléa d'une suite individuelle,
- on ne peut exprimer des propriétés que sous la forme de leurs espérances[1] dans le cadre de processus aléatoire.
  Par exemple, on pourrait exprimer les probabilités conditionnelles du bit b_i suivant connaissant le bit b_{i-1} précédent, et vérifier l'espérance de ces probabilités sur la chaîne.
  Plus la chaîne est longue, plus l'erreur sur l'estimation est faible, permettant ainsi de vérifier qu'elle respecte bien les propriétés attendues.

2.3 Approche de la théorie de l'information

En théorie de l'information, si un variable aléatoire X avec n réalisations possibles $\{s_1, \ldots, s_n\}$, chaque symbole s_i avec une probabilité p_i, alors l'entropie de X est :

$$E[X] = -\sum_{i=1}^{n} p_i \log_2 p_i$$

dont l'unité est en bits.

Alors, la théorie de l'information nous dit que toute chaîne produite par une réalisation de X de taille n peut être codée avec $n\,E[X]$ bits.

Cette limite peut effectivement être atteinte en utilisant des techniques de compression pour une chaîne assez longue.

Par exemple, pour $p_0 = \frac{1}{4}$ et $p_1 = 1 - p_0 = \frac{3}{4}$, alors son entropie est :

$$E[X] = -\left(\tfrac{1}{4}\log_2 \tfrac{1}{4} + \tfrac{3}{4}\log_2 \tfrac{3}{4}\right) \simeq 0.81 \text{ bits}$$

Donc pour $n = 1.000.000$ symboles, toute chaîne pourra être codé avec $n\,E[X] = 811.278$ bits.

Supposons maintenant que la chaîne ne soit constituée que de 1. Sa représentation peut donc être constitué du caractère répété, suivi du nombre de répétitions (codage de type runlength). Le codage de n utilise 20 bits + 1 bit pour le caractère répété, donc 21 bits.

En conséquence, si une chaîne possède une certaine forme de régularité, alors sa représentation peut être bien plus courte que celle obtenue avec la théorie de l'information.

L'entropie représente donc seulement le nombre de bits nécessaires au codage d'une chaîne quelconque dont les bits suivent une loi particulière. Elle n'est donc pas nom plus adaptée aux suites individuelles.

2.4 Caractérisation d'une suite aléatoire infinie

Une première caractérisation (par von Mises, 1919) d'une suite **aléatoire** binaire infinie $b_1 b_2 \ldots$ est la suivante :

1. soit f_n le nombre de 1 parmi les n premiers bits de la suite. Alors $\lim_{n \to \infty} \frac{f_n}{n} = p$.
2. Une règle de sélection de place est une fonction partielle ϕ qui sélectionne un sous-ensemble de bits de la suite. Le bit b_n est sélectionné si et seulement si $\phi(b_1 \ldots b_n) = 1$. La fonction retourne $\perp$ sinon.
   Notons n_i l'indice du $i^{\text{ème}}$ bit sélectionné de cette manière.
3. Alors la suite est aléatoire si pour toute règle de sélection de place, la fonction f_n tends aussi vers p pour toutes les sous-suites $b_{n_1} b_{n_2} \ldots$ obtenues par sélection.

On lui donne le nom de **collectif**.

1. **Exemple :** espérance de la valeur d'une variable aléatoire = sa moyenne.

Exemple 26: Soit la chaîne 10101010101010101 . . .
On a $f_n = 0.5$. Considérons les deux règles de sélection :

1. $\phi_1(b_1 \dots b_n) = 1$ si $n\%2 = 0$ et $\perp$ sinon.
   La sous-suite obtenue est 0000000 . . . et $f_n = 0$.
2. $\phi_2(b_1 \dots b_n) = 1$ si $n\%2 = 1$ et $\perp$ sinon.
   La sous-suite obtenue est 1111111 . . . et $f_n = 1$.

En conséquence, cette chaîne n'est pas aléatoire.

Autre façon de voir : si la suite est le résultat d'un jeu de pile-ou-face, alors un joueur qui miserait toujours sur pile aurait un gain nul, même en choisissant n'importe quelle sous-suite.

Il s'avère que cette définition n'est pas assez rigoureusement définie, et qu'elle induit qu'aucun collectif n'existe (*i.e.* aucune chaîne n'est aléatoire).

Exemples 27:

- Si on définit $\phi_1(b_1 \dots b_n) = 1$ si $b_n = 1$ et $\perp$ sinon, alors $f_n = 1$.
- Si on définit $\phi_1(b_1 \dots b_n) = 1$ si $b_n = 0$ et $\perp$ sinon, alors $f_n = 0$.

Wald (1930) démontre qu'en restreignant les ϕ admissibles à un ensemble dénombrable de fonctions, alors des collectifs existent. Mais, il ne précise pas l'ensemble de fonctions à utiliser.

Church (1940) propose de choisir l'ensemble des fonctions calculables. La définition devient alors rigoureuse. Il montre alors que, avec sa définition, non seulement les suites aléatoires existent, mais qu'elles sont abondantes.

Pour $p = \frac{1}{2}$, elles forment un ensemble de mesure 1. Autrement dit, les chaînes non aléatoires sont extrêmement rares (aussi rare que les nombres entiers parmi les nombres réels).

Remarque 17: nombres normaux

> Toutes les suites de von Mises-Wald-Church sont des nombres normaux. À savoir le nombre de 0 et de 1 sont asymptotiquement égaux (= pour toute suite assez longue, ce qui est induit par $f_n \to \frac{1}{2}$) sur tout bloc de toute longueur (induit par la règle de sélection).
>
> Mais le critère de normalité des suites est insuffisant. Le nombre de Champernowne construit de la manière suivante :
> 12345678910111213141516171819020212223242526 . . .
> est un nombre normal si on considère les répartitions de $\{0, \dots, 9\}$, alors que le $i^{\text{ème}}$ chiffre de cette suite peut facilement être calculé.

Malheureusement, cette définition ne suffit pas. Ville (1940) montre qu'il existe des suites aléatoire au sens de von Mises-Wald-Church, avec $p = \frac{1}{2}$, mais telles que $\frac{f_n}{n} \leq \frac{1}{2}$ pour tout n.

Autrement dit, si ces suites particulières sont le résultat d'un pile-ou-face, alors un joueur misant toujours sur pile accumulerait des gains. Elles ne sont donc pas aléatoires.

Le problème de fond est que des tests effectifs (règles de sélection + f_n) échouent à détecter **toutes les formes de régularité**.

Cela ne signifie pas que la régularité de certaines suites peut être prouvée, mais qu'il est impossible

de calculer effectivement le critère de von Mises-Wald-Church.

La solution est apportée un peu plus tard par Martin-Löf (1966).

Définition III.2 (test de Martin-Löf)

Un test de Martin-Löf est une suite calculable d'ensembles $\{U_n\}_{n\in\mathbb{N}}$ telle que :

- chaque U_n est un ensemble de chaînes finies.
- il existe un algorithme qui liste tous les éléments de U_n (=un énumérateur).
  x représente **toutes les chaînes qui commencent** par x.
- les ensembles U_n sont imbriqués ($U_{n+1} \subseteq U_n$).
- la mesure de U_n est bornée par 2^{-n} (à savoir $\mu(U_n) = \sum_{x\in U_n} 2^{-|x|} \leq 2^{-n}$).

La dernière condition assure que les ensembles U_n deviennent de plus en plus petit. Par exemple, si U_n contient toutes les chaînes de longueur $n+1$ commençant par 0 (donc $U_n = \{0\Sigma^n\}$), alors sa mesure est $\mu(U_n) = 2^n \cdot 2^{-(n+1)} = 2^{-1} = \frac{1}{2}$ car il y a 2^n chaînes de ce type.

Définition III.3 (passage du test de Martin-Löf)

- une suite x échoue au test de Martin-Löf si x est dans l'intersection de tous les U_n (*i.e.* $x \in \bigcap_n U_n$).
- une suite x réussi le test de Martin-Löf si x n'échoue à aucun test.

Si x est dans l'intersection de tous les U_n,

- chaque ensemble U_n contient une version partielle de x.
- tous les bits de x sont calculables.

Autrement dit, **il n'existe aucun programme capable de générer une suite aléatoire infinie**.

En revanche, pour toute suite aléatoire x de taille finie, il existe trivialement un programme qui génère x.

Exemple 28: Soit la chaîne x tirée aléatoirement 100111011101011100100110. Alors le fonction calculable qui génère x est :

```
x='100111011101011100100110'
print(x)
```

2.5 Complexité linéaire d'une suite binaire finie

Signalons pour finir l'algorithme de Massey-Berkelamp.

Les registres à décalage à rétroaction linéaire (Linear Feedback Shift Register = LFSR) constituent un modèle permettant d'estimer la complexité linéaire d'une suite binaire. Un LFSR de longueur N est une suite $\{s_i\}$ définie par :

- ses coefficients $c = (c_0, c_1, \ldots, c_{N-1})$ où $c_i \in \mathbb{B}$,
- son état initial $s = (s_0, s_1, \ldots, s_{N-1})$ où $s_i \in \mathbb{B}$
- les termes suivants $i > N$ sont calculés par la fonction récursive : $s_{i+1} = (c_0.s_i + c_1.s_{i-1} + \ldots + c_{N-1}.s_{i-(N-1)}) \mod 2$

On définit la complexité linéaire d'une suite binaire $(x_0, \ldots, x_{n-1})$ de longueur n comme étant la longueur N du plus petit LFSR telle qu'il existe des coefficients et un état initial permettant de générer la totalité de la suite.

L'algorithme de Massey-Berkelamp est un algorithme simple permettant de déterminer en temps linéaire les coefficients du plus petit LFSR générant une suite binaire finie.

Un LFSR de longueur N peut produire des suites de longueur $2^N - 1$ avant de se répéter.

3 Introduction à la complexité de Kolmogorov

3.1 Définition

Kolmogorov (1960)[2] définit une nouvelle notion de la complexité d'une chaîne.

Définition III.4 (Complexité de Kolmogorov)

La complexité de Kolmogorov d'un objet x (notée $K(x)$) est la longueur du plus petit programme qui produit la sortie x.
Autrement dit :

$$K(x) = \min_{\langle P\rangle \in \Pi} \{|\langle P\rangle| \text{ tel que } U(\langle P\rangle, \varepsilon) = x\}$$

où Π est l'ensemble des programmes P possibles, U est la machine de Turing universelle optimale, et ϵ est la chaîne vide.

On entend par $U(\langle P\rangle, \varepsilon) = x$ que la simulation de l'exécution de P sur la machine de Turing universelle avec pour entrée la chaîne vide s'arrête avec x sur sa bande.

3.2 Invariance

Mais pourquoi donner une définition avec une machine de Turing et non pas mon langage de programmation préféré ?

On rappelle que :

- Tout algorithme peut être exécuté sur une machine de Turing,
- Un langage de programmation est Turing-complet s'il peut simuler une machine de Turing. Notons $U_L(\langle M\rangle)$ le simulateur d'une machine de Turing Universelle avec le langage L.

Mais remarquons que l'inverse est également vrai : une machine de Turing peut simuler tout langage de programmation Turing-complet.

Notons $T_L(\langle P_L\rangle)$ la fonction qui transforme l'algorithme du programme P_L écrit avec langage L en son équivalent sur une machine de Turing.

En conséquence, pour tout couple de langage de programmation (L_1, L_2) Turing-complet, on peut simuler :

- avec le langage L_1 tout programme P_{L_2} avec $U_{L_1}(T_{L_2}(P_{L_2}))$,
- avec le langage L_2 tout programme P_{L_1} avec $U_{L_2}(T_{L_1}(P_{L_1}))$,

2. voir aussi Solomonoff (1960), Chaitin (1969)

Par la suite, on appelle $U_{L_j}(T_{L_i}(P))$ un interpréteur du langage L_i en langage L_j, et on le note $I_{i \to j}$.
La complexité de Kolmogorov dépend du langage utilisé, mais d'une manière bien particulière.
On a le résultat suivant ;

Théorème III.1 (invariance)

Pour tout couple de langages L_1 et L_2, il existe une constante c telle que :
$$\forall x, \left|K_{L_1}(x) - K_{L_2}(x)\right| \leq c$$

A savoir, la constante c ne dépend pas de x.

Démonstration:

Notons $I_{i \to j}$ l'interpréteur qui permet de simuler le langage L_i avec un langage L_j.
Mais, soit $P_{\min}(L_i, x)$ le plus petit programme qui produit x avec le langage L_i.
Avec ces notations, on a : $K_{L_i}(x) = |P_{\min}(L_i, x)|$.
Quelle est la différence de taille entre $K_{L_1}(x)$ et $I_{1 \to 2}(P_{\min}(L_1, x))$?

Dans le langage L_2, $I_{1 \to 2}(P_{\min}(L_1, x))$ est constitué de la taille de l'interpréteur et de la taille de $P_{\min}(L_1, x)$, à savoir $K_{L_1}(x)$.
Donc :
$|I_{1 \to 2}(P_{\min}(L_1, x))| = |I_{1 \to 2}| + K_{L_1}(x)$.
Remarquons que $|I_{1 \to 2}|$ ne dépend pas de x.
Nécessairement, comme $K_{L_i}(x)$ est la taille minimale pour le code qui produit x dans le langage L_i, il borne inférieurement la version interprétée du code minimal pour les autres langages.

- $K_{L_2}(x) \leq |I_{1 \to 2}| + K_{L_1}(x)$,
- $K_{L_1}(x) \leq |I_{2 \to 1}| + K_{L_2}(x)$

D'où, en choisissant $c = \max(|I_{1 \to 2}|, |I_{2 \to 1}|)$,
$$\forall x, \left|K_{L_1}(x) - K_{L_2}(x)\right| \leq c.$$

□

En conséquence, la complexité de Kolmogorov exprimée dans deux langages différents est la même à une constante fixe près.

3.3 Premiers exemples

Le but de cette section est de majorer la complexité de Kolmogorov à partir des codes qu'il est possible d'écrire afin d'obtenir le résultat souhaité, et non de produire le code minimal associé.

Les exemple seront dans un premier temps donné en Python. On affichera la valeur de x plutôt que de la retourner.

Exemple 29: $K(x)$
Soit une chaîne x quelconque (par exemple $x =' 10100111011100110101110 10'$).
Peut-on donner une borne supérieure de son code minimal ?
Considérons le code Python suivant :

```
x='10100111011100110101110 10'
print(x)
```

Il existe peut-être une façon plus courte de construire x, mais en l'absence d'information, on est sûr que pour n'importe quel x :

$\forall x, K(x) \leq |x| + c$
La pire représentation (au sens du code minimal) de x est x lui-même.
La constante c représente la longueur de tous les autres caractères du programme qui ne contiennent pas la représentation de x.

Exemple 30: $K(xx)$ Sachant que $\forall x, K(x) \leq |x| + c$, peut-on obtenir une borne supérieure de $K(xx)$?

```
x='10100111011100110101110 10'
x.append(x)
print(x)
```

On a donc $\forall x, K(xx) \leq |x| + c'$.
La constante c' est différente car le code n'est pas le même, mais la complexité de Kolmogorov de xx est bornée par celle de x.
Autrement dit, xx n'est pas plus complexe que x.

Exemple 31: $K(x^n)$ Et si on concatène n fois x ?

```
x='10100111011100110101110 10'
n=4392
out=''
for i in range(n): out.append(x)
print(out)
```

La partie variable du code est :

- x lui-même,
- n un nombre entier, dont la longueur en binaire est $\log_2(n)$.

Tout le reste est du code constant (on note c sa longueur).
En conséquence,
$K(x^n) \leq |x| + O(\log_2(n)) + c$.
Mais peut-on obtenir une borne encore plus fine ?

Oui ! Parce que la valeur de n peut éventuellement être elle-aussi construite et nécessiter moins de bits que sa propre description.

Exemple 32: $K(x^n)$ **et** $n = 2^p$
On peut obtenir x en doublant de manière itérative p fois sa longueur afin d'obtenir une chaîne de longueur 2^p.

```
# ici le code qui construit x dans a
out=a
p=43
for i in range(p): out.append(out)
print(out)
```

Le code minimal peut maintenant ne stocker que p à la place de 2^p.
D'où : $K(x^n) \leq K(x) + O(\log_2(p))$. Le coût de stockage de la répétition passe donc de $\log_2 2^p = p$ et $\log_2 \log_2 2^p = \log_2 p$.

On peut généraliser le résultat précédent en incluant aussi le coût de construction de p, à savoir : $K(x^n) \leq K(x) + K(p) + c$

3.4 Machine de Turing universelle optimale

Malheureusement, le code Python ne permet pas d'analyser finement ce qu'il se passe en terme de taille du code dès que les exemples deviennent un tout petit peu moins évidents. Par exemple, il est impossible d'évaluer le coût sur la taille du code d'un appel fonctionnel.

Il faut alors s'intéresser à machine de Turing universelle optimale utilisée pour évaluer la taille des codes minimaux.

Mais qu'entend-on par machine de Turing universelle optimale dans le cadre de la complexité de Kolmogorov ?

C'est une MTU U, avec :

- **une bande d'entrée :** bande en lecture seule contenant l'entrée de la MTU.
- **une bande de travail :** bande en lecture/écriture utilisable lors de l'exécution du programme.
- **une bande de sortie :** bande en écriture seule qui contient, lorsque la MTU s'arrête, la valeur retournée.

Ce type de machine est aussi utilisé dans le chapitre 5 (voir la notion de transducteur).

On adopte alors la stratégie suivante pour la MTU optimale U :

- cas d'un seul bloc :
  - un programme M est codé $10\langle M\rangle$: U exécute $\langle M\rangle(\varepsilon)$.
  - une chaîne x est codée $01\langle x\rangle$: U recopie x sur la bande de sortie.
- cas de plusieurs blocs :
  - un bloc suivi d'un autre bloc est codé $\ell(|\langle B_i\rangle|)\langle B_i\rangle$ où $\ell(|\langle B_i\rangle|)$ est le codage de la longueur de B_i (permet de lire les bits du bloc B_i, et de savoir où commence le bloc suivant).
  - le tout dernier bloc est codé sans sa longueur ($\langle B_k\rangle$) car on sait où il commence (fin du bloc précédent) et la fin du bloc est marquée par un blanc.

L'encodage de la longueur utilise le code auto-délimitant suivant : chaque bit est doublé, et la fin est marquée par 01 pour un paramètre, et de 10 pour un programme. Par exemple, pour coder un paramètre de longueur 5, on a : $\ell(5) = \ell(101_2) = 11\,00\,11\,01$. La longueur du codage d'un nombre binaire b est donc $|\ell(b)| = 2|n| + 2$.

Ainsi, si l'encodage du programme P commence par 10, c'est un programme sans paramètre. S'il commence par 01, c'est directement la chaîne de sortie. S'il commence par 00 ou 11, c'est le début du codage de la longueur du bloc qui suit. Ce type de codage est dit préfixe.

Exemples 33: de codage d'entrée pour la MTU optimale U

- $01\langle x\rangle$ = la chaîne x est directement recopiée en sortie.
- $10\langle M\rangle$ = le programme à exécuter est $M(\varepsilon)$
- $\ell\,(|M|)\,\langle M\rangle p$ = le programme à exécuter est $M(p)$
- $\ell\,(|M|)\,\langle M\rangle\langle F\rangle$ = le programme à exécuter est $M(\varepsilon)$ où M fait appel à la fonction F dans son code.

- $\ell(|M|)\langle M\rangle\ell(|F|)\langle F\rangle p$ = le programme à exécuter est $M(p)$ où M fait appel à la fonction F dans son code.
- ...

Un appel de fonction pour une MTU se déroule ainsi :

- On place le curseur en fin de bande de travail avec un marqueur de début de bande.
- On exécute la fonction qui poursuit l'écriture en fin de bande de sortie.
- A la fin de l'exécution, la portion de bande de travail utilisée par la fonction est effacée.

Regardons le codage de l'entrée d'une MTU universelle dans les cas déjà étudiés.

Exercice 21. *Donner le code minimal à placer en entrée de la MTU optimale pour les cas suivants. Puis donner une borne supérieure de $K(x)$.*

1. *$K(x)$ où x est une chaîne quelconque.*
2. *$K(x)$ où on connaît une fonction f_x qui construit x et telle que $|f_x| \ll |x|$.*
3. *$K(xx)$ où x est une chaîne quelconque.*
4. *$K(xx)$ où on connaît une fonction f_x qui construit x et telle que $|f_x| \ll |x|$.*
5. *En déduire une majoration de $K(xx)$ si on connaît le code minimal pour f_x.*
6. *$K(x^n)$ où x est une chaîne quelconque.*
7. *$K(x^n)$ où on connaît une fonction f_x qui construit x (avec $|f_x| \ll |x|$).*
8. *$K(x^n)$ où on connaît une fonction f_x qui construit x (avec $|f_x| \ll |x|$), et une fonction f_n qui construit n (avec $|f_n| \ll |n|$)*
9. *En déduire une majoration de $K(x^n)$ si on connaît le code minimal pour f_x et f_n.*
10. *La formule de Bellard permet de calculer $i^{ème}$ décimale de π pour tout i. Évaluer le complexité de Kolmogorov des n premières décimales de π.*

Exemple 34: $K(xy)$ Il faut écrire un code qui produit la concaténation de x et y. Ce code contient un code f_x qui génère x et le recopie sur la bande de sortie, et f_y qui procède de manière identique pour y.
Un code pour xy pour la MTU optimale est :
$\langle P\rangle = \ell(|\langle M\rangle|)\langle M\rangle\ell(|\langle f_x\rangle|)\langle f_x\rangle\langle f_y\rangle$
où M est le programme (de taille constante) ci-dessous :
$M(\varepsilon) = \begin{array}{|l} f_x(\varepsilon) \\ f_y(\varepsilon) \end{array}$
La taille de M est une constante c (indépendante de $|f_x|$ et de $|f_x|$).
Ainsi, $K(xy) \leq |f_x| + |f_y| + O(\log_2 |\langle f_x\rangle|)$.
Notons que la constant c disparaît dans $O(\log_2 |\langle f_x\rangle|)$
Mais, remarquons que la place de f_x et de f_y peut être intervertie dans le codage. Une écriture alternative de l'encodage de P est :
$\langle P'\rangle = \ell(|\langle M\rangle|)\langle M\rangle\ell\left(\left|\langle f_y\rangle\right|\right)\langle f_y\rangle\langle f_x\rangle$
De façon similaire $K(xy) \leq |f_x| + |f_y| + O(\log_2 |\langle f_y\rangle|)$.

Comme on veut une borne supérieure du code minimal, le meilleur encodage de P est celui tel que f_x et f_y sont les codes minimaux, en codant en premier le code le plus court.
$K(xy) \leq K(x) + K(y) + O(\log_2 \min(K(x), K(y)))$.

On notera que la finesse de la borne précédente est impossible à obtenir avec un code Python, car celui-ci ne permet pas de prendre en compte la longueur effective du codage.

3.5 Information

Cette nouvelle notion de complexité lui permet alors de donner la définition suivante :

Définition III.5 (information)
La quantité d'information dans une chaîne x est $K(x)$.

La théorie de l'information qui donnait une borne minimale pour toutes les chaînes, y compris celles aléatoires.

Nous avons ainsi un moyen de définir la quantité d'information dans une chaîne individuelle.

4 Compression

4.1 Lien avec Kolmogorov

La définition de la complexité de Kolmogorov donne l'impression qu'elle pourrait y avoir un lien avec la compression.

```
y=# une version compressée de x
# code de décompression de y dans a
print(a)
```

Si `y` est la compression maximale de `x`, est-ce que ce code ne serait pas en mesure de produire le code de longueur minimale qui construit `x` ?

Clairement, on a : $K(x) \leq |y| + c$ où c est le code de décompression de y.

Pour répondre à cette question, il est nécessaire de faire une brève introduction à la compression. Les parallèles avec la complexité de Kolmogorov apparaîtront au fur et à mesure.

Définissons tout d'abord les termes du problème.

4.2 Qu'est ce que la compression ?

Qu'est ce que la compression ?
C'est la science de la représentation de l'information sous une forme compacte.

Définition III.6 (compression sans perte)
Soit C une fonction de compression et C^{-1} la fonction de décompression. Alors C est une fonction de compression sans perte si $\forall x \in \mathbb{B}^*$, $C^{-1}(C(x)) = x$.

Dans le cadre de ce cours, on considérera que la compression sans perte.

Qu'est-ce-qu'une forme compacte ?
C'est une forme dans laquelle on a tenté de réduire au maximum la redondance, en la codant d'une manière particulière.

Qu'est ce que l'information ?
Vaste question, mais nous donnons une première réponse qui peut sembler circulaire : c'est ce qui ne peut pas être compressé.
Principe de la compression : Il s'agit pour une suite de bits de départ :

1. de trouver une suite alternative de bits avec un algorithme de compression,
2. tel que cette suite alternative occupe une place de stockage moins importante que la suite initiale,
3. et qu'il soit possible de reconstituer la suite de bits initiale à partir de cette suite alternative avec l'algorithme de décompression associé.

Intuitivement, un algorithme de compression dépend des données à compresser (la redondance dans un texte n'est pas de même nature que celle issue d'une suite de mesures sur un capteur).

4.3 Digression philosophique

En terme métaphysique,

- le langage permet d'exprimer une représentation compressée de la réalité ou de la pensée,
- les mathématiques sont une représentation symbolique compressée d'une pensée logique,
- la physique consiste à "compresser" la réalité en un ensemble d'équation permettant de la décrire,
- ...

Sur le fond, l'intelligence humaine fonctionne :

- en structurant le réel ou les idées sous forme de concept (symboles),
- en compressant le réel ou les idées sous forme d'une description de celui-ci à partir des concepts (suite de symboles),

Autrement dit, nous compressons l'information dans notre cerveau en :

- faisant sens de ce que nous voulons compresser,
- ne retenant et n'organisant que les informations qui nous semblent significatives.

4.4 Définition d'un compresseur

Mais avant de répondre à cette question, nous devons tout d'abord définir ce qu'est un compresseur.

On rappelle les notations utilisées :

$\mathbb{B}^*$ l'ensemble des mots binaires.
$\mathbb{B}^n$ l'ensemble des mots binaires de longueur n exactement.
$|x|$ la longueur du mot x.

Définition III.7 (compresseur)

Un compresseur est une fonction C injective de $\mathbb{B}^* \to \mathbb{B}^*$ qui possède au moins un mot u tel que $|C(u)| \leq |u|$.

Si $|C(u)| < |u|$, la compression est dite stricte.

Remarques 18:

- Un injection est une application telle que $C(u) = C(v) \Rightarrow u = v$.
  Si deux mots différents sont compressés, alors leurs compressions est différentes.
- Autre conséquence, si un mot u peut être compressé par F, alors $C(u)$ est inversible, et $C^{-1}(C(u)) = u$.
  Attention, cela ne signifie pas que pour tout v, $C^{-1}(v)$ existe (*i.e.* toute chaîne aléatoire de bits ne représente pas nécessairement le résultat d'une compression par C).

Cette définition est vraiment peu contraignante : elle demande juste à ce que le compresseur n'augmente pas la longueur d'au moins un mot. On aimerait évidemment qu'un compresseur soit capable d'effectuer une compression stricte sur un sous-ensemble non négligeable de $\mathbb{B}^*$.

4.5 Premières propriétés

Proposition III.1 (nombre de mots inférieur à n bits)

Il y a moins de mots de taille strictement inférieure à n que de mots de taille n.

Démonstration:

L'ensemble $\mathbb{B}^{<n} = \cup_{i=1}^{n-1} \mathbb{B}^i$ des mots de taille strictement inférieure à n a pour cardinal $\#\mathbb{B}^{<n} = \sum_{i=1}^{n-1} \#\mathbb{B}^i$ (car les $\mathbb{B}^i$ sont disjoints).
D'où $\#\mathbb{B}^{<n} = 2 + 2^2 + \cdots + 2^{n-1} = \frac{2^n-1}{2-1} - 1 = 2^n - 2$.
On a donc bien $\#\mathbb{B}^n > \#\mathbb{B}^{<n}$ car $2^n > 2^n - 2$. □

Regardons comment est structuré l'ensemble des mots compressibles et compressés :

- si on note $E = \mathbb{B}^*$ et l'image $F = \mathbb{B}^*$. Le compresseur est une fonction de E dans F.
- comme $E = F$, chacun de ces ensembles a 2 éléments de 1 bit, 4 éléments de 2 bit, 8 éléments de 3 bits, …, 2^n éléments de n bits, …

Supposons que le compresseur C ne compresse qu'un seul élément $u \in E$, alors il y a nécessairement un élément qui est dilaté.

En effet, si un u est compressé ($|C(u)| < |u|$),

- comme il n'y a que $2^{|C(u)|}$ éléments de taille $|C(u)|$, au moins un élément v de $\mathbb{B}^{|C(u)|}$ a été soit compressé, soit dilaté.
- s'il est dilaté, alors on a montré qu'au moins un élément a été dilaté.
- s'il est compressé, alors on reprend récursivement l'argument avec v.

Tout se passe comme un jeu de chaises musicales : le nombre de chaises auxquelles correspondent des mots binaires de n bits ou moins est très limité. Donc, tout élément u telle que $|u| > n$ et $|C(u)| < n$, oblige au moins un élément quelconque de ne plus avoir de place.

Que se passe-t-il si on applique itérative un compresseur sur un mot déjà compressé ?

Proposition III.2

Pour n'importe quel mot de départ sur lequel on applique de manière répétée un compresseur, on se trouve nécessairement dans l'un des deux cas suivants :

1. soit on est dans un cycle (= suite finie de mots qui se répète indéfiniment),
2. soit les mots successifs obtenus deviennent arbitrairement grands.

Démonstration:

Soit C la méthode de compression. Notons $C^n(u)$ l'application de n compressions successives C sur le mot u et $\mathbb{C}(u) = \cup_{k>0} C^k(u)$ l'ensemble contenant toutes ces compressions successives de u.
Deux cas sont alors possibles :

- soit $\mathbb{C}(u)$ est fini, et dans ce cas $C^n(u)$ cycle nécessairement entre les valeurs de $\mathbb{C}(u)$ (puisqu'il y a une infinité dénombrable de $C^n(u)$).
  On notera que si $\exists p$ tel que $C^p(u) = u$, alors on entre dans un cycle de longueur p et $\#\mathbb{C}(u) = p$.
- soit $\mathbb{C}(u)$ n'est pas fini, mais comme tout $\mathbb{B}_{<p}$ est fini, si on compresse plus de $\#\mathbb{B}_{<p}$ fois, alors on trouvera nécessairement un $C^n(u)$ tel que $|C^n(u)| > p$, et ceci pour toute valeur de p.

□

Proposition III.3 (principe des tiroirs de Dirichlet)

Soit E et F deux ensembles **finis** tels que $\#E > \#F$. Alors, il n'existe pas d'application injective de E dans F.

Démonstration:

Si E est un ensemble de n paires de chaussettes, F un ensemble de p tiroirs, avec $n > p$, alors il est impossible de ranger toutes les paires de chaussettes dans les tiroirs si on ne met qu'une seule paires par tiroir.
La démonstration mathématique est équivalente. □

Proposition III.4 (rareté des mots compressibles)

La proportion de mots de taille n qui sont compressibles d'au moins p bits par une méthode de compression C quelconque est strictement inférieur à 1 pour 2^{p-1}.

Démonstration:

Au mieux, C est une injection de $\mathbb{B}^n$ dans $\mathbb{B}^{\leq n-p}$ (compression de mots de taille n vers n'importe quel mot de taille $\leq n-p$).
Par la proposition *III*.1, on sait que $\#\mathbb{B}^{<n-p+1} < 2^{n-p+1}$.
Donc, par le principe des tiroirs de Dirichlet, seuls 2^{n-p+1} peuvent au mieux être compressé parmi les $\#\mathbb{B}^n = 2^n$; soit une fraction de 1 pour $2^n/2^{n-p+1} = 2^{p-1}$. □

Exemple 35: Soit les chaînes de longueur $n = 1024$ bits. La proportion de chaînes de longueur n compressible de 10% est donc de 1 pour $2^{102-1} \simeq 2,5 \times 10^{30}$.

4.6 Conséquences

Les propriétés démontrées précédemment ont les conséquences suivantes :

- l'application itéré d'un compresseur sur un mot d'entrée
  - soit a une borne minimale (cyclage : c'est le plus petit mot du cycle),
  - soit dilate le mot indéfiniment.
- un compresseur universel (à savoir, un compresseur qui compresserait tout mot) n'existe pas. Il y a moins de mots de taille inférieure à n que de mots de taille n.

- le nombre de mots compressibles est très faible.
  La fraction du nombre de mots compressibles de p bits diminue **au moins** exponentiellement en 2^{p-1}.
  En réalité, c'est encore beaucoup moins (la démonstration ne compressait que les mots de n bits).

Comme le nombre de mots compressible de manière appréciable est très faible, est-il intéressant de compresser ?

Exercice 22. *Soit C un algorithme de compression sans perte. Notons $C(x)$ comme le résultat de l'exécution de C sur x et $C^n(x) = \underbrace{C(C(C(\dots C}_{n\,fois}(x))))$ la compression itérée n fois. Par ailleurs, si $C(x) > |x|$, alors C ne compresse pas.*

1. *Qui signifie que C est un algorithme de compression sans perte et interpréter $C(x) > |x|$? On se souviendra de la proposition III.1.*
2. *La compression itérée a-t-elle un intérêt ?*
3. *Donnez une borne inférieure sur la longueur de $C^n(x)$ en fonction de n, x et d'une constante indépendante des deux.*

Évidemment, oui car :

- l'humain produit naturellement de l'information redondante (langue orale, langue écrite, ...)
- le stockage d'information dans la mémoire d'un ordinateur est :
  — guidé avant tout par la rapidité d'accès à l'information (alignement).
  — exceptionnellement peu efficace (exemple : **ü** apparaît dans 4 mots du dictionnaire sur 100000 mots, alors qu'il est codé sur le même nombre de bits que le **e** qui a une fréquence d'apparition d'environ 15%).
  — il existe des ensembles de méthodes de compression différentes adaptées aux types de données que l'on souhaite compresser.
- évidemment, lorsqu'un compresseur C est appliqué sur une chaîne u et que $|C(u)| > |u|$, alors u n'est pas compressé (sortie $0u$ si u n'est pas compressé, et $1u$ si u ne l'est pas. Coût=1 bit).

4.7 Pour aller plus loin

Pour compléter et donner une idée sur la façon dont les principales méthodes de compression fonctionnent :

- le codage entropique qui utilise la fréquence individuelle des symboles (codage de Huffman, codage Arithmétique, ...),
- le codage par dictionnaire qui construit un dictionnaire de motifs (LZ77, LZ78, LZW, ...),
- le codage prédictif qui utilise les probabilités conditionnelles pour prédire les symboles suivants à partir du contexte (Partial Prédiction Matching, Dynamic Markov Coding, ...)

Un logiciel de compression utilise un mélange de ces techniques compresser des données. La méthode utilisée dépend également du type des données.

Deux références pour aller plus loin :

- **Data compression : the complete reference** de David Salomon,
- **Théorie des codes : compression, cryptage, correction** de Jean-Guillaume Dumas.

5 Propriété de la complexité de Kolmogorov

5.1 Calculabilité

Théorème III.2 (calculabilité de $K(x)$)

$K(x)$ n'est pas calculable.

Démonstration: par l'absurde

Supposons que $K(x)$ soit calculable.
Considérons la machine de Turing suivante :

$M(\varepsilon) =$
```
POUR tout x ∈ Σ*
    SI K(x) > |⟨M⟩|
    ALORS RETOURNER x
```

Il existe des programmes minimaux de toute taille (voir le lemme III.2 plus loin).
Donc, on finit toujours par trouver un x vérifiant $K(x) > |\langle M \rangle|$.
Mais M produit la sortie x, et sa taille est inférieure à $K(x)$.
Donc $K(x)$ ne produit pas un résultat correct, et elle n'est pas calculable. □

Une preuve similaire a été effectuée pour le problème des MTs minimales en utilisant le théorème de récursion.

5.2 Incompressibilité

Proposition III.5

Considérons l'ensemble des chaînes binaires de longueur n.
La fraction des chaînes x de longueur n telles que $K(x) < n - k$ est inférieure à 2^k.

Démonstration:

On sait que $K(x) \leq |x| + c$ où $c \ll |x|$. Autrement dit, le code minimal pour produire x n'est pas plus grand que la longueur de x, à une constante c près dépendante du langage et négligeable devant x quand celui-ci est grand.
Dans cette démonstration, $\mathbb{B}^n$ est l'ensemble des codes minimaux de longueur n. Comme déjà démontré dans la partie compression (voir la proposition III.1), $\#\mathbb{B}^{<n-k} < 2^{n-k}$.
Cela signifie qu'il y a moins de 2^{n-k} codes de longueur $n - k$.
Or, il y a 2^n chaînes de longueur n.
En conséquence, parmi ces chaînes, seules 2^{n-k} sont susceptibles d'avoir un code minimal de longueur inférieure à $n - k$ (principes des tiroir de Dirichlet, proposition III.3).
Donc, la fraction des chaînes de longueur n telles que $K(x) < n - k$ est de $2^n/2^{n-k} = 2^k$. □

Lemme III.1

Pour tout $n > 1$, il existe une chaîne incompressible de longueur n.

Démonstration:

On a vu que $\mathbb{B}^{<n} < \mathbb{B}^n$ (il existe moins de mots de longueurs strictement inférieures à n que de mots de longueurs n).
En conséquence, il existe moins de codes minimaux de longueurs strictement inférieure à n qui de code chaîne de longueur n

Par le principes des tiroirs de Dirichlet (proposition III.3), il existe des chaînes dont le code minimal est de longueur n.
Ceci est vrai pour tout $n > 1$. □

Il existe donc des chaînes incompressibles de toutes tailles.
Puis le lemme utilisé dans la démonstration que la complexité de Kolmogorov n'est pas calculable.

Lemme III.2
Il existe une constante c, telle que $\forall n \geq c$, il existe un code minimal de longueur n.

Démonstration:
On a vu dans l'introduction sur la complexité de Kolmogorov qui la machine de Turing universelle optimale U associée était en mesure de produire x avec l'appel $U(00x)$.
En conséquence, si x est une chaîne non compressible de longueur $K(x) = |x| + 2$.
En prenant $c = 2$, n'importe quelle chaîne incompressible x de longueur $n - 2$ a bien un code minimal de longueur n. □

Ce lemme démontre donc qu'il existe donc des codes minimaux de toutes tailles supérieure à 2.

Exercice 23.

1. *Soit x et y deux chaînes incompressibles. Montrer que* $K(x) + K(y) > K(xy) + O(1)$.
2. *Soit les chaînes de la forme* $s = 0^n y$ *construite avec n 0 consécutifs suivis d'une chaîne y. Montrer que l'on peut choisir un* $c = O(1)$ *tel que qu'il existe des chaînes de la forme s qui sont incompressible.*
3. *En déduire que pour tout c, il existe des chaînes x et y pour lesquelles* $K(xy) > K(x) + K(y) + c$.

Exercice 24. *Montrer que l'ensemble des chaînes incompressibles ne contient aucun sous-ensemble infini récursivement énumérable.*
Ligne de preuve : *supposer qu'un tel ensemble A existe, donc il existe une MT M qui reconnaît A (attention M peut boucler). Construire alors un énumérateur* $N(k)$ *qui retourne la première chaîne de A de longueur au moins k (attention, il doit être décidable). Calculer la* $K(N(k))$ *et conclure.*

Théorème III.3 (indécidabilité de la compressibilité)
Soit l'ensemble des chaînes incompressibles :
$$L = \{x \in \Sigma^* \mid K(x) \geq |x|\}$$
L est indécidable.

Démonstration: par l'absurde
Supposons que L soit décidable. Alors il existe un décideur R qui décide L (*i.e.* $R(x)$ accepte si $x \in L$ et rejette si $x \notin L$).
Mais si un tel décideur existe, alors $K(x)$ devient calculable, à une constante près, pour tous les x tels que $R(x)$ accepte.
Notons qu'on peut avoir une idée précise de cette constante, puisque le code minimal associé à une chaîne incompressible ne fait que la retourner.
Or $K(x)$ n'est pas calculable (voir théorème III.2). Donc, R n'existe pas, et L n'est pas décidable. □

Comme l'ensemble des chaînes compressible est le complémentaire de L, il n'est pas décidable non plus.

5.3 Lien avec la notion d'aléatoire

Définition III.8 (Aléa de Kolmogorov)

Les chaînes x telles que $K(x) \geq |x|$ sont aléatoires au sens de Kolmogorov.

Pour résumer,

- chaîne incompressible = chaîne aléatoire.
- chaîne compressible = chaîne non aléatoire.

Remarque 19:

Pour tout x, le code minimal est incompressible, sinon il serait possible de trouver un code plus court en le compressant.

Comme l'information que nous générons est naturellement redondante, une chaîne incompressible est donc, soit aléatoire, soit de l'information compressée.

Remarque 20:

Soit une suite infinie $s = s_1 s_2 \ldots$. On note $s^n = s_1 \ldots s_n$ les n premiers symboles de cette suite.

Les deux notions suivantes sont **équivalentes** :

- une chaîne aléatoire au sens de Martin-Löf : à savoir une suite infinie s qui passe tous les tests de Martin-Löf,
- une chaîne aléatoire qui a la complexité de Kolmogorov [3] maximale pour tous les segments initiaux finis ($\forall n, \exists c, K(s^n) \geq n - c$).

5.4 Comportement de la complexité de Kolmogorov

Quel est le comportement de $K(x)$?
Prenons $x \in \mathbb{N}$ (ce qui est équivalent à considérer son écriture en binaire).

- $K(x)$ est bornée inférieurement par une fonction croissante
  exemple : $K(1^n) \geq \log_2(n)$
- $K(x) \simeq \log_2(x)$ presque partout
  car la plupart des chaînes sont incompressibles
- $K(x)$ décroche infiniment souvent
  si x est compressible, alors $K(x) \simeq K(xx) \simeq K(xxx) \simeq \ldots$, $K(x) \simeq K(x^R)$, ... à savoir pour toutes transformations de x par un code de taille fixe.
- $K(x)$ a une certaine forme de continuité.
  $\forall x, |K(x) - K(x \pm 1)| < c$ (une fois x obtenu, il est facile de lui ajouter une constante avec un code de taille fixe).

5.5 Autre applications

La complexité de Kolmogorov permet dans certains cas des démonstrations surprenantes de résultats bien connus. Notamment,

- la non régularité de $0^n 1^n$ (normalement avec lemme de l'étoile),
- le théorème d'Euclide (nombre infini de nombres premiers),
- ...

Proposition III.6

$L = \{0^n 1^n \mid n \in \mathbb{N}\}$ n'est pas régulier.

Démonstration:

Supposons que L soit régulier. Alors il existe un ADF A qui le décide.
En conséquence, pour toute chaîne $x \in L$, $K(x) \leq |\langle A \rangle| + c$.
À savoir, toute expression régulière peut être décidée avec un code de taille finie, et sa complexité de Kolmogorov peut être bornée par la longueur description de A et du code qui simule A. La somme de ces deux longueur **est une constante**.
Supposons maintenant que $x = 0^n 1^n$. Alors nécessairement, la valeur de n doit être calculée par le code minimal (comptage du nombre de 0, et vérification du nombre de 1).
Or, le stockage de n dans ce code minimal occupe $\log_2 n$ bits, ce qui implique qu'il vérifie : $K(0^n 1^n) \geq \log_2 n$.
Donc, $\log_2 n \leq K(x) \leq |\langle A \rangle| + c$
Mais on peut faire en sorte que $\log_2 n > |\langle A \rangle| + c$, ce qui contredit l'hypothèse que L est régulier. □

Théorème III.4 (Euclide)

Il existe une infinité de nombres premiers.

Démonstration: (par l'absurde)

Supposons qu'il y ait un nombre k fini de nombres premiers.
Soit x une chaîne incompressible, et n la représentation entière de x.
Alors il existe $\{e_1, \ldots, e_k\}$ tels que $n = p_1^{e_1} \ldots p_k^{e_k}$ (sa décomposition en facteurs premiers).
Nécessairement $e_i \leq \log_2 n$. Donc $|e_i| \leq \log_2 \log_2 n$ (par exemple, si $n = 2^k$ alors $p_1 = 2$, $e_1 = \log_2 n = k$, et $|e_1| = \log_2 \log 2^k = \log k$). L'inégalité est stricte pour $p_i > 2$.
Son code minimal contient les k nombres premiers (constants), et la représentation de tout n a besoin au plus de k entiers (les e_k), chacun étant inférieur à $\log_2 \log_2 n$.
Donc, $K(x) < k \log_2 \log_2 n + c$.
Ceci contredit le fait que x soit incompressible. Donc, l'hypothèse est fausse et le nombre de nombres premiers n'est pas fini. □