Cours 3 : Algorithmes de codage de source

30 septembre 2016

Plan du cours

1. Retour sur Huffman ; optimalité du code de Huffman ;
2. Codage à base d'intervalles : code de Shannon-Fano-Elias et code de Shannon ;
3. Codage arithmétique.

1. Codage et décodage à l'aide d'un code préfixe

Codage : Accès à une table indexée par les lettres.

Décodage : Parcours dans l'arbre du code.

- On part de la racine de l'arbre.
- À chaque bit lu de la séquence à décoder on descend à droite ou à gauche suivant sa valeur.
- Lorsque l'on atteint une feuille, on obtient une lettre du message et on retourne à la racine.

Code optimal

Les théorèmes de Kraft et Mac Millan ont un corollaire immédiat :

Corollaire 1. *S'il existe un code à décodage unique dont les K mots ont pour longueur $n_1, n_2, \ldots, n_K$ alors il existe un code préfixe avec les mêmes longueurs.*

$$\sum_{k=1}^{K} \frac{1}{2^{n_k}} \leq 1.$$

Définition Un code à décodage unique d'une source X est *optimal* s'il n'existe pas de code à décodage unique de X ayant une longueur moyenne strictement inférieure.

Proposition 1. *Pour toute source il existe un code préfixe optimal.*

Code de Huffman

Soit la source discrète X d'alphabet $\mathcal{X} = \{a_1, \ldots, a_{K-2}, a_{K-1}, a_K\}$ munie de la loi p. Sans perdre en généralité, nous pouvons supposer que $p(a_1) \geq \ldots \geq p(a_{K-1}) \geq p(a_K) > 0$.

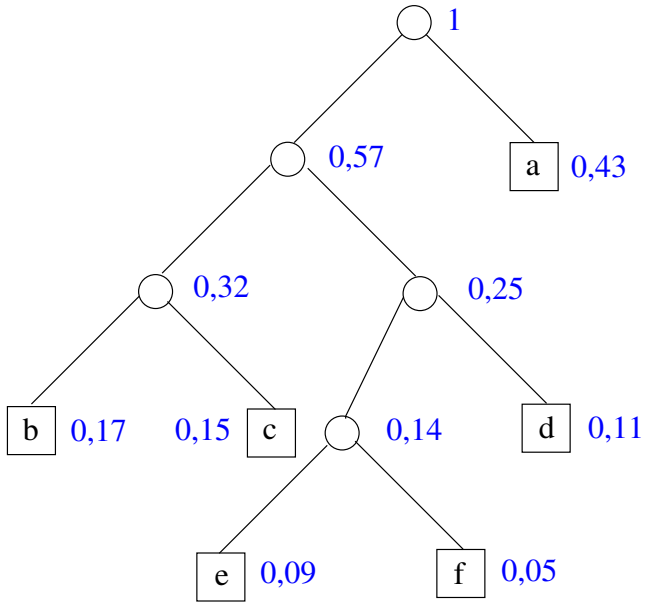
Soit Y d'alphabet $\mathcal{Y} = \{a_1, \ldots, a_{K-2}, b_{K-1}\}$ munie de la loi

$$\begin{array}{rcll} q(a_k) & = & p(a_k), & k = 1 \ldots, K-2 \\ q(b_{K-1}) & = & p(a_{K-1}) + p(a_K) & \end{array}$$

Algorithme (Huffman) Nous construisons récursivement φ_K un code préfixe de X. Si $K = 2$, les mots de code sont $\varphi_k(a_1) = 0$ et $\varphi_k(a_2) = 1$. Si $K > 2$, soit φ_{K-1} un code de Huffman de Y,

- $\varphi_K(a_k) = \varphi_{K-1}(a_k)$ pour $k = 1 \ldots, K-2$,
- $\varphi_K(a_{K-1}) = \varphi_{K-1}(b_{K-1}) \parallel 0$,
- $\varphi_K(a_K) = \varphi_{K-1}(b_{K-1}) \parallel 1$,

Code de Huffman : exemple



x	$P(x)$	$-\log_2(P(x))$	$\varphi(x)$	n_x
a	0.43	1.22	1	1
b	0.17	2.56	000	3
c	0.15	2.74	001	3
d	0.11	3.18	011	3
e	0.09	3.47	0100	4
f	0.05	4.32	0101	4

$H = 2.248$ $\quad \bar{n} = 2.28$

$E = 98.6\%$

Le code de Huffman est optimal

Lemme 1. *Pour toute source, il existe un code préfixe optimal tel que*

1. *si* $p(x_j) > p(x_k)$ *alors* $|\phi(x_j)| \leq |\phi(x_k)|$.
2. *Les deux mots de code les plus longs ont la même longueur, et correspondent aux deux symboles les moins probables.*
3. *Les deux mots de code les plus longs ne diffèrent qu'en leur dernier bit.*

Preuve du 1

1. si $p(x_j) > p(x_k)$ alors $|\phi(x_j)| \leq |\phi(x_k)|$.

Soit ϕ optimal et supposons $|\phi(x_j)| > |\phi(x_k)|$.

Soit ϕ' obtenu en échangeant les mots de code de x_j et x_k. Alors

$$\begin{aligned} |\phi'| - |\phi| &= \sum p(x)|\phi'(x)| - \sum p(x)|\phi(x)| \\ &= p(x_j)|\phi(x_k)| + p(x_k)|\phi(x_j)| - p(x_j)|\phi(x_j)| - p(x_k)|\phi_(x_k)| \\ &= (p(x_j) - p(x_k))(|\phi(x_k)| - |\phi(x_j)|) \\ &< 0 \end{aligned}$$

Ce qui contredit l'optimalité de ϕ.

Preuve du 2

2. Les deux mots de code les plus longs ont la même longueur.

S'ils n'ont pas la même longueur, on peut enlever le dernier bit du plus long, et préserver la propriété du préfixe.

$\Longrightarrow$ on obtient un code plus court.

De plus ce sont les deux symboles les moins probables.

Preuve du 3

3. Les deux mots de codes les plus longs ne diffèrent que par leur dernier bit, et correspondent aux deux symboles les moins probables.

Chaque $\phi(x)$ de longueur maximale a un jumeau (sinon on peut le raccourcir).

Parmi ceux de longueur maximale, il y a les deux moins probables.

Après une permutation, on peut mettre les deux moins probables de sorte qu'ils soient jumeaux.

Le code de Huffman est optimal

Preuve (Récurrence sur la taille k de l'alphabet) Soit f optimal sur X avec $p(a_1) \geq \cdots \geq p(a_{k-1}) \geq p(a_k)$, avec $f(a_{k-1}) = m||0$ et $f(a_k) = m||1$. On construit g sur $Y = (a_1, \ldots, a_{k-2}, b_{k-1})$:

$$\left\{ \begin{array}{rcl} g(a_j) & = & f(a_j), \quad j \leq k-2 \\ g(b_{k-1}) & = & m \end{array} \right.$$

On calcule

$$\begin{array}{rcl} |f| - |g| & = & p(a_{k-1}) + p(a_k) \\ |\varphi_k| - |\varphi_{k-1}| & = & p(a_{k-1}) + p(a_k) \end{array}$$

Donc $|f| - |\varphi_k| = |g| - |\phi_{k-1}|$.
Optimalité de $\varphi_{k-1} \Rightarrow |g| - |\phi_{k-1}| \geq 0 \Rightarrow |f| - |\varphi_k| \geq 0$.
Optimalité de f et $|f| - |\varphi_k| \geq 0 \Rightarrow |f| - |\varphi_k| = 0 \Rightarrow$ optimalité de φ_k.

Inconvénients du codage de Huffman

- Nécessite d'avoir/de faire au préalable des stats. sur les proba. d'occurrence des symboles de l'alphabet. Si on utilise la distribution de probabilité q à la place de la vraie distribution de probabilité p, au lieu de compresser avec un taux $\approx H(p)$ on compresse avec un taux $\approx H(p) + D(p||q)$. On peut remédier à ce problème en utilisant un algorithme de codage adaptatif qui calcule les stats à la volée et les affine au fil du codage.
- Fondé sur un modèle de source discrète sans mémoire.
- Intérêt de travailler sur des regroupements de symboles de taille l assez grande :
  (1) modèle de source sans mémoire $+$ réaliste
  (2) efficacité du codage $\to 1$.
  Mais complexité en mémoire $= O(|\mathcal{X}|^l)$.

Le code de Huffman en pratique

- Apparait pratiquement partout...
  1. dans les algorithmes de compression gzip, pkzip, winzip, bzip2
  2. les images compressées jpeg, png
  3. l'audio compressée mp3

  même l'algorithme de compression de Google Brotli (fin 2014) l'utilise...

2. Codage à base d'intervalles

- Rend transparent le fait que l'on utilise pour coder le symbole $x_i \approx -\log p(x_i)$ bits et donc $|\phi| \approx H(X)$.
- Idée du codage arithmétique.

Nombres 2-adiques

Dans l'intervalle $[0, 1]$, ils sont de la forme

$$\sum_{i=1}^{\infty} d_i \, 2^{-i}, \text{ où } d_i \in \{0, 1\}$$

On écrira $0.d_1 d_2 d_3 \ldots$

Par exemple

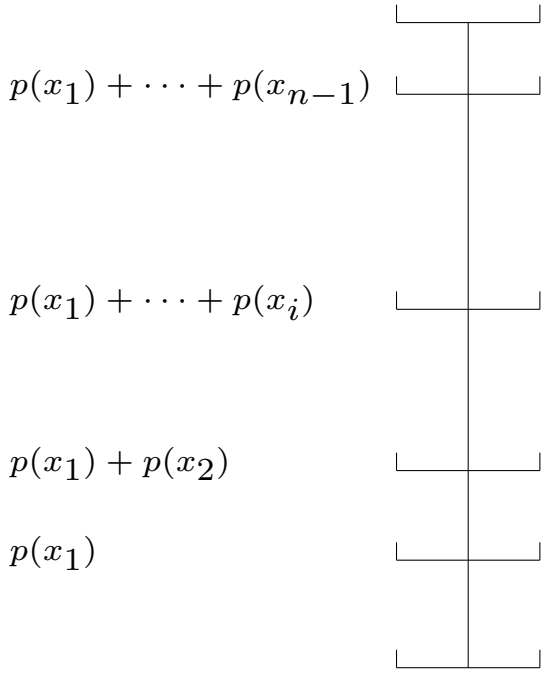
0.25	$\to$	0.01	0.43	$\to$	$0.0110111000\ldots$
0.125	$\to$	0.001	0.71	$\to$	$0.1011010111\ldots$
0.625	$\to$	0.101	$1/\sqrt{2}$	$\to$	$0.1011010100\ldots$

Certains nombres ont plusieurs développements, par exemple $0.25 \to 0.01000\ldots$ et $0.25 \to 0.00111\ldots$. Dans ce dernier cas on choisira le développement de valuation minimale (le plus court).

Code de Shannon-Fano-Elias

Soit X une source discrète d'alphabet $\mathcal{X}$, de loi de probabilité p.

$$S(x) = \sum_{x'<x} p(x')$$



La largeur de chaque intervalle est $p(x_i)$. On « code chaque x_i par son intervalle ».
(*Pour la plus petite lettre $S(x)$ vaut 0*)

Codage de Shannon-Fano-Elias

On considère le milieu de l'intervalle.

$$\bar{S}(x) \stackrel{\text{def}}{=} \frac{1}{2} p(x) + S(x).$$

Nous définissons $\varphi(x)$ pour tout $x \in \mathcal{X}$ comme les $\lceil -\log p(x) \rceil + 1$ premiers bits du développement 2-adique de $\bar{S}(x)$.

Proposition 2. *Le code φ est préfixe et sa longueur moyenne $|\varphi|$ vérifie*

$$H(X) \leq |\varphi| < H(X) + 2$$

Le principe utilisé

Lemme 2. *Pour tous réels u et v dans $[0,1[$, et pour tout entier $l > 0$, si $|u - v| \geq 2^{-l}$ alors les l premiers bits des développements 2-adiques de u et v ne sont pas tous identiques.*

Le code de Shannon-Fano-Elias est préfixe

Preuve *Sans perte en généralité, on peut supposer* $y > x$

$$\begin{aligned}
\bar{S}(y) - \bar{S}(x) &= \frac{p(y)}{2} + \sum_{x'<y} p(x') - \frac{p(x)}{2} - \sum_{x'<x} p(x') \\
&= \frac{p(y)}{2} - \frac{p(x)}{2} + \sum_{x \le x'<y} p(x') \\
&= \frac{p(y)}{2} + \frac{p(x)}{2} + \sum_{x<x'<y} p(x') > \max\left(\frac{p(y)}{2}, \frac{p(x)}{2}\right) \\
&> \max\left(2^{-\lceil -\log p(x)\rceil - 1}, 2^{-\lceil -\log p(y)\rceil - 1}\right)
\end{aligned}$$

Donc ils diffèrent sur les $\min(\lceil -\log p(x)\rceil + 1, \lceil -\log p(y)\rceil + 1)$ *premiers bits :* $\phi(x)$ *et* $\phi(y)$ *ne peuvent être préfixes l'un de l'autre.*

Code de Shannon-Fano-Elias : exemple

x	$p(x)$	$\lceil -\log p(x) \rceil + 1$	$\bar{S}(x)$		$\varphi(x)$	*Huffman*
a	0.43	3	0.215	$0.0011011\ldots$	001	0
b	0.17	4	0.515	$0.1000001\ldots$	1000	100
c	0.15	4	0.675	$0.1010110\ldots$	1010	101
d	0.11	5	0.805	$0.1100111\ldots$	11001	110
e	0.09	5	0.905	$0.1110011\ldots$	11100	1110
f	0.05	6	0.975	$0.1111100\ldots$	111110	1111

Code de Shannon-Fano-Elias : autre exemple

x	$p(x)$	$\lceil -\log p(x) \rceil + 1$	$\bar{S}(x)$		$\varphi(x)$	*Huffman*
a	0.25	3	0.125	0.001	001	10
b	0.5	2	0.5	0.1	10	0
c	0.125	4	0.8125	0.1101	1101	110
d	0.125	4	0.9375	0.1111	1111	111

Si on enlève le dernier bit à φ le code n'est plus préfixe.

x	$p(x)$	$\lceil -\log p(x) \rceil + 1$	$\bar{S}(x)$		$\varphi(x)$	*Huffman*
b	0.5	2	0.25	0.01	01	0
a	0.25	3	0.625	0.101	101	10
c	0.125	4	0.8125	0.1101	1101	110
d	0.125	4	0.9375	0.1111	1111	111

Si on enlève le dernier bit à φ le code reste préfixe.

Code de Shannon

Le code de Shannon est défini de la même manière que celui de Shannon-Fano-Elias, à deux exceptions près :
- *les lettres sont rangées par probabilités décroissantes,*
- *le mot codant x est constitué des $\lceil -\log p(x) \rceil$ premiers bits de $S(x)$.*

(En particulier, le mot codant la lettre la plus probable n'est composé que de $\lceil -\log p(x) \rceil$ '0').

Proposition 3. *Le code de Shannon est préfixe et sa longueur moyenne $|\varphi|$ vérifie*

$$H(X) \leq |\varphi| < H(X) + 1$$

Le code de Shannon est préfixe

Preuve *Pour tous* $x, y \in \mathcal{X}$ *tels que* $x < y$ *(donc* $\lceil -\log p(y) \rceil \geq \lceil -\log p(x) \rceil$*)*

$$\begin{aligned} S(y) - S(x) &= \sum_{z<y} p(z) - \sum_{z<x} p(z) \\ &= \sum_{x \leq z < y} p(z) \\ &= p(x) + \sum_{x<z<y} p(z) \geq p(x) \geq 2^{-\lceil -\log p(x) \rceil} \end{aligned}$$

Preuves sur les longueurs

***Proposition* 4.** *Le code φ de Shannon-Fano-Elias vérifie $H(X) \leq \varphi < H(X) + 2$*

Preuve

$$|\varphi(x)| = \lceil -\log p(x) \rceil + 1 < -\log_2 p(x) + 2.$$

On conclut en passant à la moyenne.

***Proposition* 5.** *Le code φ de Shannon vérifie $H(X) \leq \varphi < H(X) + 1$*

$$|\varphi(x)| = \lceil -\log p(x) \rceil < -\log_2 p(x) + 1.$$

On conclut en passant à la moyenne.

Le code de Shannon atteint l'entropie dans le cas dyadique

Dans le cas dyadique les probabilités vérifient $p(x) = 2^{-\lceil -\log p(x) \rceil}$.

Dans ce cas les longueurs du codage de Shannon vérifient

$$|\phi(x)| = \lceil -\log p(x) \rceil = -\log_2 p(x)$$

En passant à la moyenne

$$\sum p(x)|\phi(x)| = -\sum p(x) \log_2 p(x) = H(X).$$

Comme le code de Huffman il est optimal : il atteint lui aussi l'entropie dans le cas dyadique.

Non optimalité du codage de Shannon

Dans la source suivante

1. *émission de* 1 *avec la probabilité* $1 - 2^{-10}$ *;*
2. *émission de* 0 *avec la probabilité* 2^{-10}*.*

1. *Le codage de Shannon attribue à 1 un mot de longueur* $\lceil -\log(1 - 2^{-10}) \rceil = \lceil 0.0014 \rceil = 1$
2. *Le codage de Shannon attribue à 0 un mot de longueur* $\lceil -\log 2^{-10} \rceil = \lceil 10 \rceil = 10$

Le code de Huffman code avec 1 *et* 0*, ce qui est optimal.*

3. Codage arithmétique

- *Un des inconvénients majeurs du code de Huffman est sa complexité en mémoire quand on veut regrouper par paquets de taille l les lettres de l'alphabet (complexité $C = O(|\mathcal{X}|^l)$).*

$$r = \textit{ redondance } \stackrel{\text{def}}{=} 1 - \textit{ efficacité.}$$

Code de Huffman sur les blocs de taille l : $r \approx O\left(\frac{1}{l}\right) \approx O\left(\frac{1}{\log C}\right)$.

- *Le codage arithmétique permet de travailler sur des paquets de taille arbitraire à un coût algorithmique acceptable. Ici aussi*

$$r \approx O\left(\frac{1}{l}\right),$$

mais la dépendance de la complexité par rapport à l est bien meilleure.

L'idée fondamentale du codage arithmétique

Au lieu de coder les lettres de $\mathcal{X}$, on travaille directement sur l'alphabet $\mathcal{X}^l$ où l est la longueur du mot à coder.

Pour coder $x_1, \ldots, x_l$

1. *on calcule l'intervalle $[S(x_1, \ldots, x_l), S(x_1, \ldots, x_l) + p(x_1, \ldots, x_l)]$, avec $S(x_1, \ldots, x_l) \stackrel{\text{def}}{=} \sum_{(y_1,\ldots,y_l)<(x_1,\ldots,x_l)} p(y_1, \ldots, y_l)$,*
2. *on code $(x_1, \ldots, x_l)$ par un élément de l'intervalle dont la longueur du développement 2-adique est $\lceil \log(p(x_1, \ldots, x_l)) \rceil + 1$.*

Cela identifie parfaitement l'intervalle en question $\Rightarrow$ déduire $x_1 \ldots x_l$ au décodage.

Avantage du codage arithmétique

$$\textit{longueur du codage}(x_1, \ldots, x_l) = -\log p(x_1, \ldots, x_l) + O(1).$$

$\Rightarrow$ *on perd au plus* $O(1)$ *bits pour coder* l *symboles et non* $O(l)$ *avec les méthodes précédentes !*

Particulièrement avantageux sur l'exemple précédent d'une suite binaire produisant avec une probabilité 2^{-10} *des* 0*. On obtient bien une longueur de codage de l'ordre de* $h(2^{-10})l + O(1) \approx 0.011l$ *au lieu de* l *avec le code de Huffman !*

La difficulté majeure du codage arithmétique

Il faut pouvoir calculer $S(x_1, \ldots, x_l)$ *avec une précision de* $\lceil \log p(x_1, \ldots, x_l) \rceil + 1$ *bits.*

$$\Downarrow$$

pouvoir faire des calculs avec une précision arbitraire.

L'idée clé pour travailler sur des paquets de taille l

Principe : Calcul *efficace* de $p(x_1, \ldots, x_l)$ et $S(x_1, \ldots, x_l)$.

- Calcul de $p(x_1, \ldots, x_l)$

$$p(x_1, \ldots, x_l) = p(x_1) \ldots p(x_l)$$

- Calcul *itératif* de $S(x_1, \ldots, x_l)$: ordre *lexicographique* sur $\mathcal{X}^l$.

$$S(x_1, \ldots, x_l) \stackrel{\text{def}}{=} \sum_{(y_1, \ldots, y_l) < (x_1, \ldots, x_l)} p(y_1, \ldots, y_l).$$

Proposition 6. $S(x_1, \ldots, x_l) = S(x_1, \ldots, x_{l-1}) + p(x_1, \ldots, x_{l-1}) S(x_l)$.

Preuve

On note m le plus petit élément de l'alphabet pour l'ordre choisi

$$\begin{aligned} S(x_1,\ldots,x_l) &= \sum_{(y_1,\ldots,y_l)<(x_1,\ldots,x_l)} p(y_1,y_2,\ldots,y_l) \\ &= A+B \qquad \text{où} \end{aligned}$$

$$A \stackrel{\text{def}}{=} \sum_{(y_1,\ldots,y_l)<(x_1,\ldots,x_{l-1},m)} p(y_1,y_2,\ldots,y_l)$$

$$B \stackrel{\text{def}}{=} \sum_{(x_1,\ldots,x_{l-1},m)\leq(y_1,\ldots,y_l)<(x_1,\ldots,x_l)} p(y_1,y_2,\ldots,y_l)$$

Preuve (suite)

$$
\begin{aligned}
A &= \sum_{(y_1,\ldots,y_l)<(x_1,\ldots,x_{l-1},m)} p(y_1, y_2, \ldots, y_l) \\
&= \sum_{y_1,\ldots,y_l:(y_1,\ldots,y_{l-1})<(x_1,\ldots,x_{l-1})} p(y_1, y_2, \ldots, y_{l-1}) p(y_l) \\
&= \sum_{(y_1,\ldots,y_{l-1})<(x_1,\ldots,x_{l-1})} p(y_1, y_2, \ldots, y_{l-1}) \sum_l p(y_l) \\
&= \sum_{(y_1,\ldots,y_{l-1})<(x_1,\ldots,x_{l-1})} p(y_1, y_2, \ldots, y_{l-1}) \\
&= S(x_1, \ldots, x_{l-1}).
\end{aligned}
$$

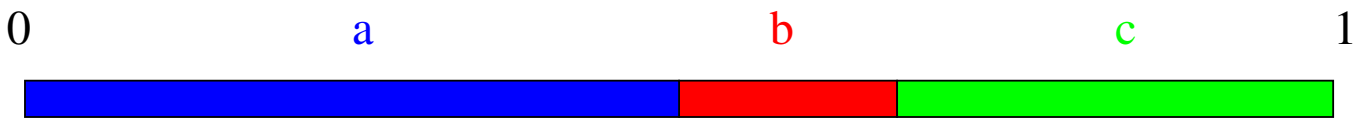
Preuve (fin)

$$
\begin{aligned}
B &= \sum_{(x_1,\ldots,x_{l-1},m)\leq(y_1,\ldots,y_l)<(x_1,\ldots,x_l)} p(y_1,y_2,\ldots,y_l) \\
&= \sum_{m\leq y_l<x_l} p(x_1,\ldots,x_{l-1},y_l) \\
&= \sum_{m\leq y_l<x_l} p(x_1,\ldots,x_{l-1})p(y_l) \\
&= p(x_1,\ldots,x_{l-1})S(x_l)
\end{aligned}
$$

Un phénomène fractal

Exemple :

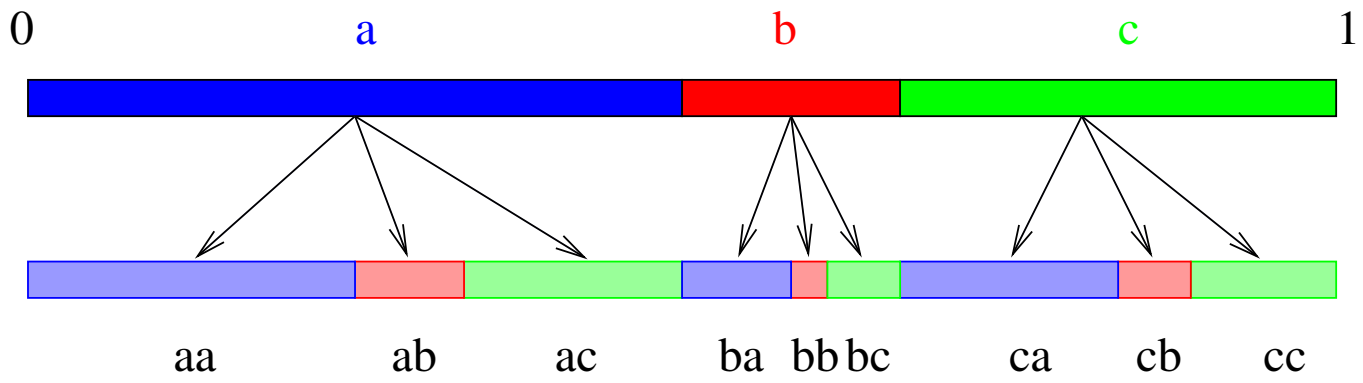
$$\mathbf{Prob}(a) = \frac{1}{2}; \mathbf{Prob}(b) = \frac{1}{6}, \mathbf{Prob}(c) = \frac{1}{3}.$$



Un phénomène fractal

Exemple :

$$\mathbf{Prob}(a) = \frac{1}{2}; \mathbf{Prob}(b) = \frac{1}{6}, \mathbf{Prob}(c) = \frac{1}{3}.$$



Procédé (de codage)

$$S(x_1,\ldots,x_l) = \underbrace{S(x_1,\ldots,x_{l-1})}_{\text{début de l'intervalle précédent}} + \underbrace{p(x_1,\ldots,x_{l-1})}_{\text{long. intervalle précédent}} S(x_l)$$

Donc si l'intervalle en cours en $[a,b[$*, et qu'on lit la lettre* x_i*, on fait*

$$\begin{aligned} a_{\text{nouveau}} &\leftarrow a + (b-a)S(x_i) \\ b_{\text{nouveau}} &\leftarrow a_{\text{nouveau}} + (b-a)p(x_i) \end{aligned}$$

La largeur de l'intervalle devient $(b-a)p(x_i) = p(x_i)\ldots p(x_{i-1})p(x_i) = p(x_1,\ldots,x_i)$.

Zooms

On travaille sur l'intervalle $[a, b[$

- *Si* $b \leq 1/2$, $\quad [a, b[\to [2a, 2b[$, *on peut sortir un bit (“0”) ;*

- *Si* $a \geq 1/2$, $\quad [a, b[\to [2a - 1, 2b - 1[$, *on peut sortir un bit (“1”)*

- *Si* $1/4 \leq a < b \leq 3/4$, $\quad [a, b[\to [2a - 1/2, 2b - 1/2[$, *on ne sort pas de bits (underflow expansion), mais on s'en rappelle (augmentation d'un compteur).*

Décodage

Soit $r = \varphi(x_1, \ldots, x_l)$ le nombre réel dont le développement 2-adique est le mot codant $(x_1, \ldots, x_l) \in \mathcal{X}^l$. Les lettres $x_1, \ldots, x_l$ sont les seules vérifiant

$$\begin{array}{rcccl} S(x_1) & < & r & < & S(x_1) + p(x_1) \\ S(x_1, x_2) & < & r & < & S(x_1, x_2) + p(x_1, x_2) \\ & & \vdots & & \\ S(x_1, \ldots, x_l) & < & r & < & S(x_1, \ldots, x_l) + p(x_1, \ldots, x_l) \end{array}$$

Procédé

$$S(x_1) \quad < \quad r \quad < \quad S(x_1) + p(x_1)$$

Supposons x_1 déterminé. On a

$$\begin{aligned}
S(x_1, x_2) < \quad & r & < S(x_1, x_2) + p(x_1, x_2) \\
S(x_1) + p(x_1)S(x_2) \leq \quad & r & < S(x_1) + p(x_1)S(x_2) + p(x_1)p(x_2) \\
p(x_1)S(x_2) \leq \quad & r - S(x_1) & < p(x_1)(S(x_2) + p(x_2)) \\
S(x_2) \leq \quad & \frac{r - S(x_1)}{p(x_1)} & < S(x_2) + p(x_2)
\end{aligned}$$

On recherche donc dans quel intervalle tombe $r_1 = \frac{r - S(x_1)}{p(x_1)}$.