Lecture 3 : Algorithms for source coding

Outline

1. Huffman code; proof of optimality;
2. Coding with intervals: Shannon-Fano-Elias code and Shannon code;
3. Arithmetic coding.

1. Coding and decoding with a prefix code

Coding : Using a table indexed by the letters.

Decoding : Using the associated binary tree

- Start at the root.
- For each bit turn left or right.
- When a leaf is attained, the corresponding letter is output and go back to the root.

Optimal Code

Corollary of Kraft and Mac Millan theorems:

Corollary 1. *If there exists a uniquely decidable code with K words of length $n_1, n_2, \ldots, n_K$ then there exists a prefix code with the same lengths.*

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

Définition A uniquely decodable code of a source X is *optimal* if there exists no other uniquely decodable code with a smaller average length.

Proposition 1. *For every source there exists an optimal prefix code.*

Huffman Code

Let X be a discrete source with alphabet $\mathcal{X} = \{a_1, \ldots, a_{K-2}, a_{K-1}, a_K\}$ with prob. distr. p. W.l.o.g. we can assume that $p(a_1) \geq \ldots \geq p(a_{K-1}) \geq p(a_K) > 0$.

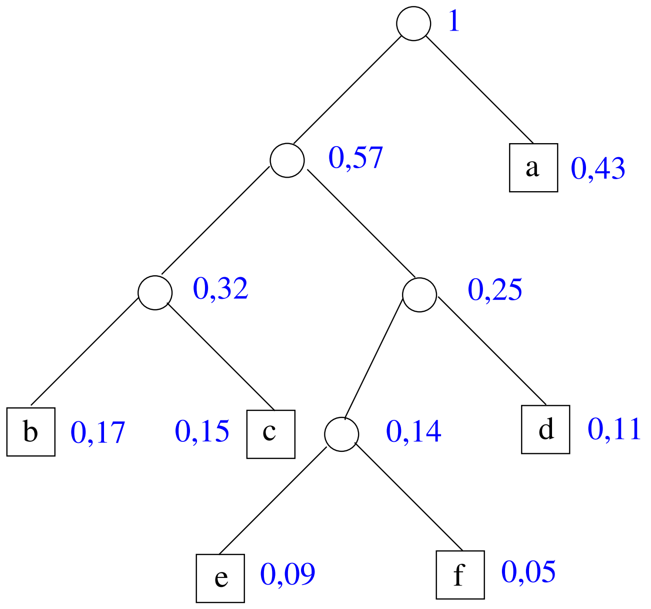
Let Y be the source with alphabet $\mathcal{Y} = \{a_1, \ldots, a_{K-2}, b_{K-1}\}$ and prob. distr.

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

Algorithm (Huffman) We compute recursively a prefix code of X. When $K = 2$, the words are given by $\varphi_K(a_1) = 0$ and $\varphi_K(a_2) = 1$. If $K > 2$, let φ_{K-1} be a Huffman code for Y,

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

Huffman Code : example

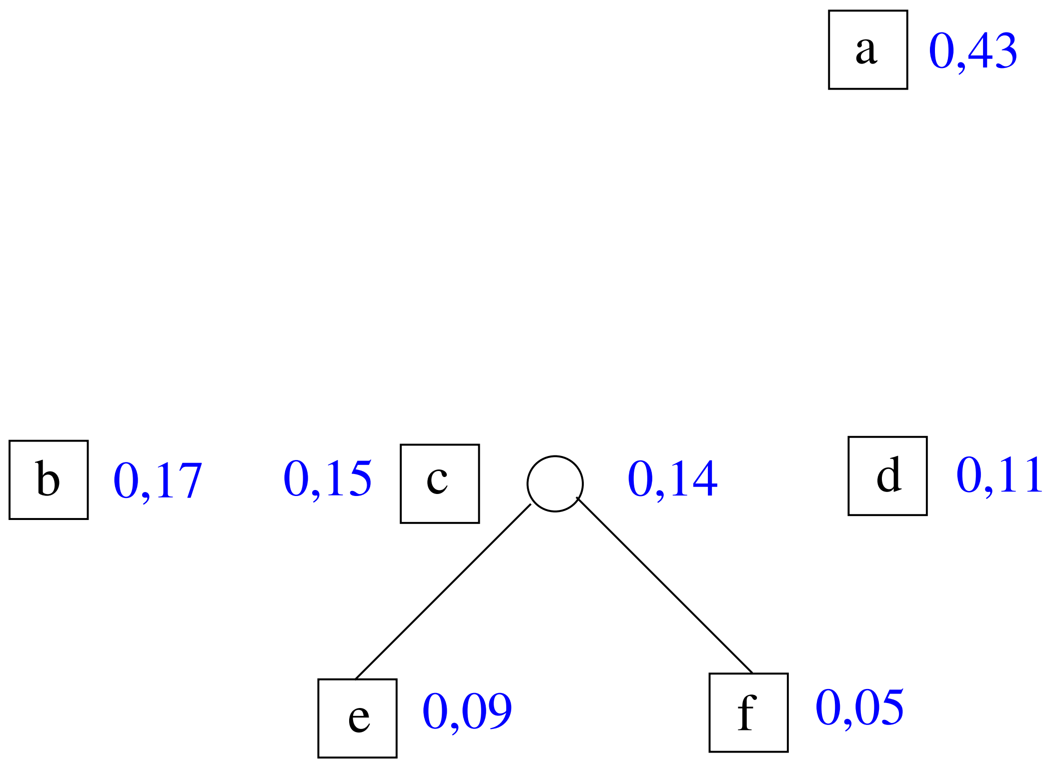


x	$P(x)$	$-\log_2(P(x))$	$\varphi(x)$	n_x
a	0.43	1.22	1	1
b	0.17	2.56	000	3
c	0.15	2.74	001	3
d	0.11	3.18	011	3
e	0.09	3.47	0100	4
f	0.05	4.32	0101	4

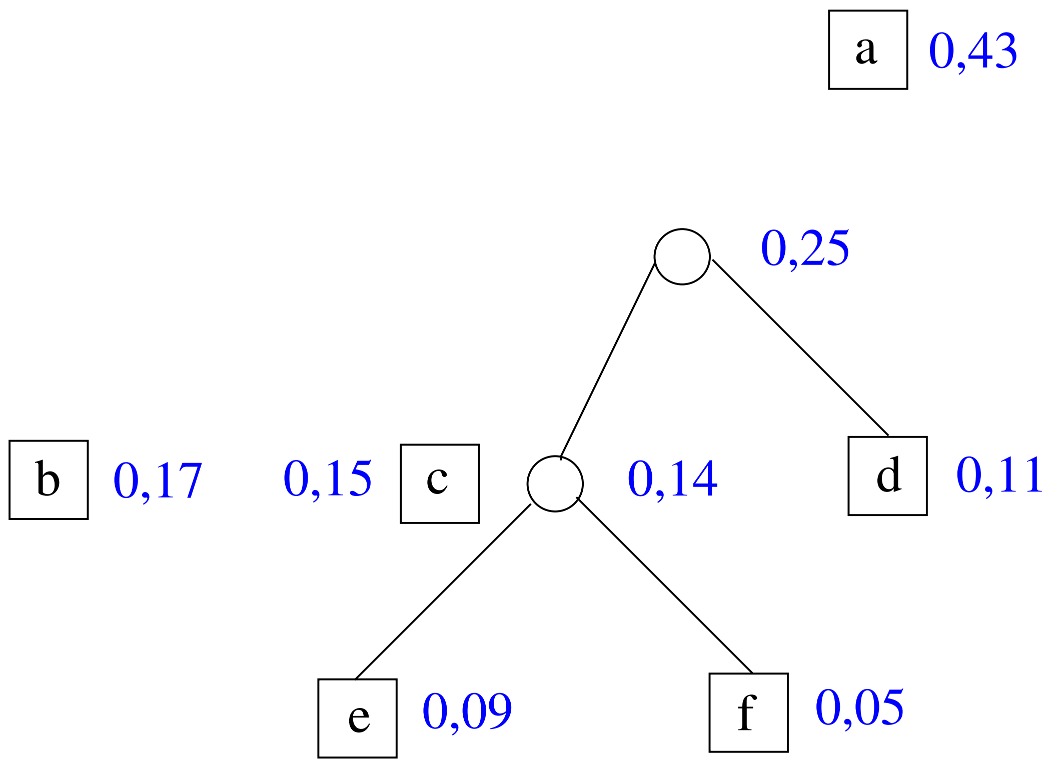
$H = 2.248$ $\bar{n} = 2.28$

$E = 98.6\%$

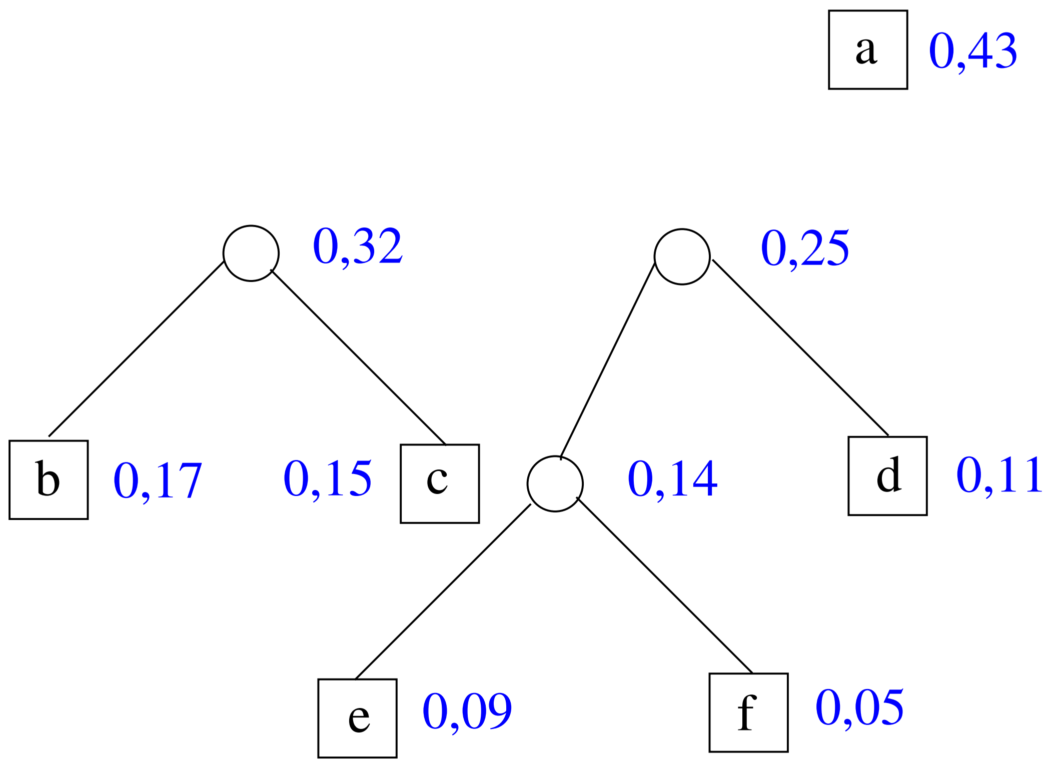
Huffman code : example (I)



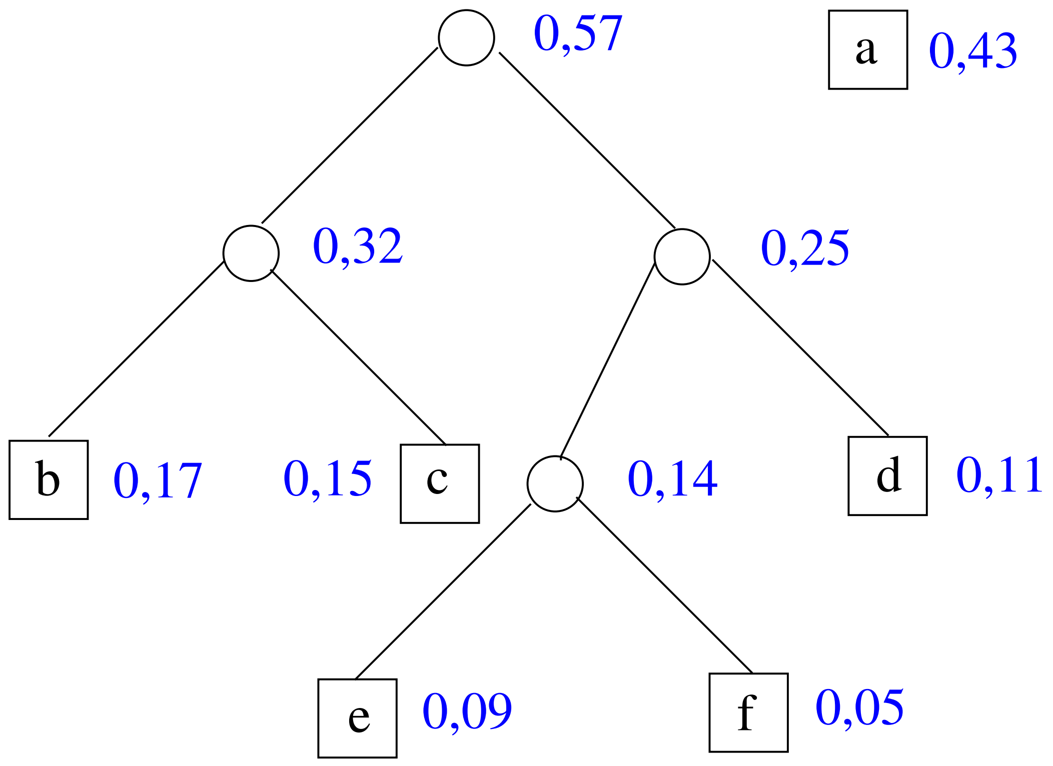
Huffman code : example (II)



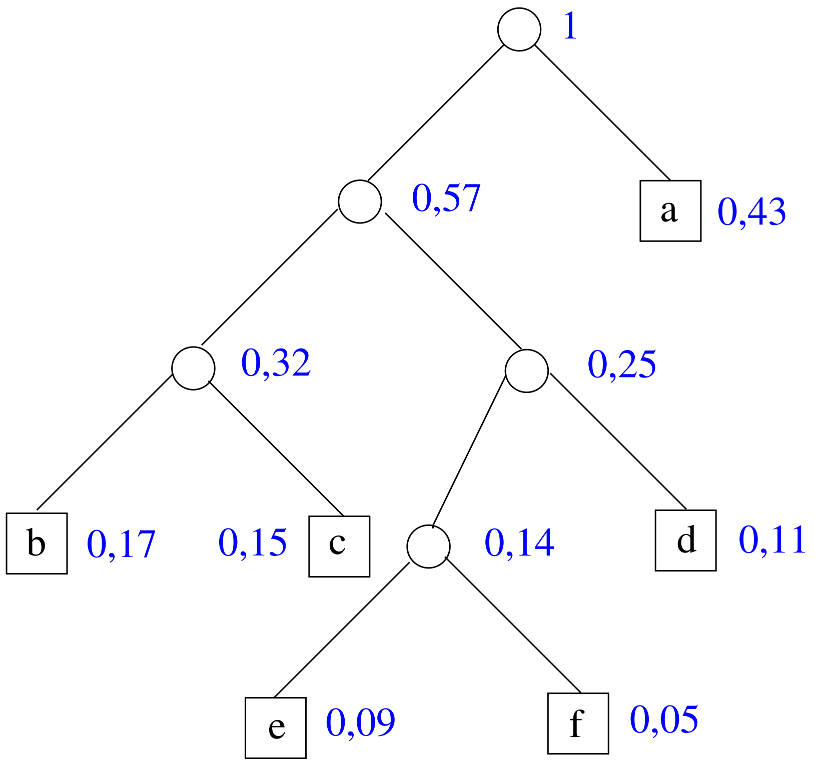
Huffman code : example (III)



Huffman code : example (IV)



Huffman code : example (last step)



The Huffman code is optimal

Lemma 1. *For every source, there exists an optimal prefix code such that*

1. *if* $p(x_j) > p(x_k)$ *then* $|\phi(x_j)| \leq |\phi(x_k)|$.
2. *The two longest codewords have the same length and correspond to the least likely symbols.*
3. *These two codewords differ only in their last bit.*

Proof of 1

1. If $p(x_j) > p(x_k)$ then $|\phi(x_j)| \leq |\phi(x_k)|$.

Let ϕ be an optimal code and assume that $|\phi(x_j)| > |\phi(x_k)|$.

ϕ' is obtained by swapping the codewords of x_j and x_k. Then

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

This contradicts the optimality of ϕ.

Proof of 2

2. The two longest codewords have the same length.

If they do not have the same length then a bit can be removed from the longest one- the resulting code is still a prefix code.

$\Longrightarrow$ a shorter prefix code is obtained in this way.

By the previous point we know that they correspond to the least likely symbols.

Proof of 3.

3. The two longest codewords differ only in their last bit.

Every $\phi(x)$ of maximal length has a brother (otherwise it would be possible to shorten the code).

Among those, there are two which correspond to the least likely symbols. By swapping the leaves we can arrange these two symbols so that they become brothers.

The Huffman code is optimal

Proof (Induction on k the alphabet size) Let f be optimal for X with $p(a_1) \geq \cdots \geq p(a_{k-1}) \geq p(a_k)$, $f(a_{k-1}) = m||0$ and $f(a_k) = m||1$. We derive from f a code g on $Y = (a_1, \ldots, a_{k-2}, b_{k-1})$:

$$\begin{cases} g(a_j) &= f(a_j), \quad j \leq k-2 \\ g(b_{k-1}) &= m \end{cases}$$

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

Hence $|f| - |\varphi_k| = |g| - |\phi_{k-1}|$.
Optimality of $\varphi_{k-1} \Rightarrow |g| - |\phi_{k-1}| \geq 0 \Rightarrow |f| - |\varphi_k| \geq 0$.
Optimality of f and $|f| - |\varphi_k| \geq 0 \Rightarrow |f| - |\varphi_k| = 0 \Rightarrow$ optimality of φ_k.

Drawbacks of the Huffman code

- Need to know beforehand the source statistics. If the wrong probability distribution q instead of the true distribution p :

$$H(X) + D(p||q) \leq |\phi| \leq H(X) + D(p||q) + 1.$$

  This can be dealt with by using an adaptative algorithm that computes the statistics on the fly and updates them during the coding process.

- Based on a memoryless source model.
- It is better to group letters in blocks of large size l: (1) memoryless model sufficiently realistic
  (2) coding efficiency $\to 1$.
  But space complexity $= O(|\mathcal{X}|^l)$.

2. Interval coding

- Gives a scheme for which we have in a natural way $|x_i| \approx -\log p(x_i)$ bits and therefore $|\phi| \approx H(X)$.
- Idea of arithmetic coding.

2-adic numbers

In the interval $[0, 1]$, they are of the form

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

We write $0.d_1 d_2 d_3 \ldots$

For instance

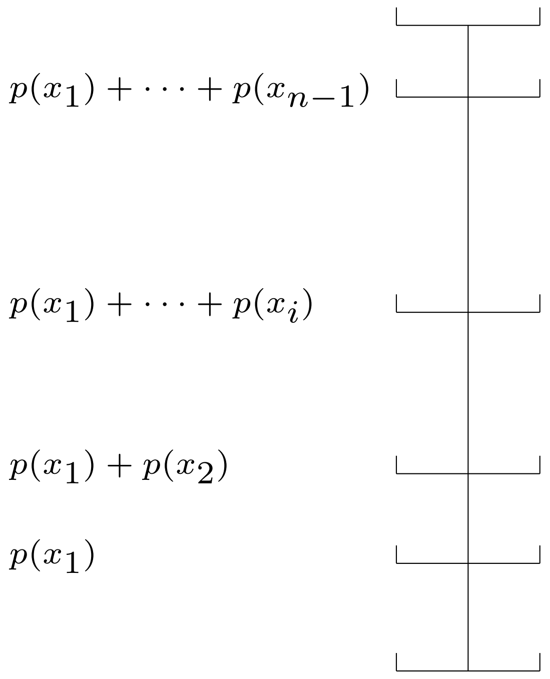
0.25	$\rightarrow$	0.01	0.43	$\rightarrow$	$0.0110111000\ldots$
0.125	$\rightarrow$	0.001	0.71	$\rightarrow$	$0.1011010111\ldots$
0.625	$\rightarrow$	0.101	$1/\sqrt{2}$	$\rightarrow$	$0.1011010100\ldots$

Certain numbers have several different 2-adic representation, for instance $0.25 \rightarrow 0.01$ and $0.25 \rightarrow 0.00111\ldots$. In the last case we choose the finite representation.

Shannon-Fano-Elias Code

Let X be a memoryless source with alphabet $\mathcal{X}$, and probability distribution p.

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



The interval width is $p(x_i)$. Each x_i is encoded by its interval.

Shannon-Fano-Elias Coding

Consider the middle of the interval

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

Let $\varphi(x)$ for all $x \in \mathcal{X}$ be defined as the $\lceil -\log p(x) \rceil + 1$ first bits of the 2-adic representation of $\bar{S}(x)$.

Proposition 2. *The code φ is prefix and its average length $|\varphi|$ verifies*

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

A simple explanation

Lemma 2. *For all reals u and v in $[0,1[$, and for every integer $l > 0$,if $|u-v| \geq 2^{-l}$ then the l first bits of the 2-adic representation of u and v are distinct.*

The Shannon-Fano-Elias code is prefix

Proof *W.l.o.g., we may assume $y > x$*

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

Therefore they differ in their $\min(\lceil -\log p(x) \rceil + 1, \lceil -\log p(y) \rceil + 1)$ first bits: $\phi(x)$ et $\phi(y)$ can not be prefix of each other.

Shannon-Fano-Elias Code : example

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

Shannon-Fano-Elias code : another example

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

If the last bit of φ is deleted the code is not prefix anymore.

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

If the last bit of φ is deleted the code is still a prefix code.

Shannon Code

The Shannon code is defined in the same way as the Shannon-Fano-Elias code, with the exception of :

- *The symbols are ordered by decreasing probability,*
- *The codeword of x is formed by the $\lceil -\log p(x) \rceil$ first bits of $S(x)$.*

(In particular, the codeword corresponding to the most likely letter is formed by $\lceil -\log p(x) \rceil$ '0').

Proposition 3. *The Shannon code is prefix and its average length $|\varphi|$ verifies*

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

The Shannon code is prefix

Proof *For all $x, y \in \mathcal{X}$ such that $x < y$ (therefore $\lceil -\log p(y) \rceil \geq \lceil -\log p(x) \rceil$)*

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

Proof for the lengths

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

Proof

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

The conclusion follows by taking the average.

Proposition 5. *The Shannon code* φ *verifies* $H(X) \leq \varphi < H(X) + 1$

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

The conclusion follows by taking the average.

The Shannon code is optimal for a dyadic distribution

In the dyadic case, the probabilities verify $p(x) = 2^{-\lceil -\log p(x) \rceil}$.

In this case the lengths of encoding satisfy

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

By taking the average

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

As for the Huffman code : average coding length $=$ *entropy.*

Non optimality of Shannon's code

For the following source

1. *output* 1 *with probability* $1 - 2^{-10}$*;*
2. *output* 0 *with probability* 2^{-10}*.*

1. *The Shannon code encodes 1 with a codeword of length* $\lceil -\log(1 - 2^{-10}) \rceil = \lceil 0.0014 \rceil = 1$
2. *The Shannon code encodes 0 with a codeword of length* $\lceil -\log 2^{-10} \rceil = \lceil 10 \rceil = 10$

The Huffman code uses 1 *and* 0*, instead which is of course optimal.*

3. Arithmetic coding

- *One of the main drawbacks of the Huffman code is the space complexity when packets of length l are formed for the alphabet letters (space complexity $C = O(|\mathcal{X}|^l)$).*

$$r = \text{ redundancy} \overset{\text{def}}{=} 1 - \text{efficiency}.$$

Huffman code on the blocks of size l: $r \approx O\left(\frac{1}{l}\right) \approx O\left(\frac{1}{\log C}\right)$.

- *Arithmetic coding allows to work on packets of arbitrary sizes with acceptable algorithmic cost. Here*

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

but the dependency of the space complexity in l is much better.

The fundamental idea

Instead of encoding the symbols of $\mathcal{X}$, work directly on $\mathcal{X}^l$ where l is the length of the word which is to encoded. .

To encode $x_1, \dots, x_l$

1. *compute the interval $[S(x_1, \dots, x_l), S(x_1, \dots, x_l) + p(x_1, \dots, x_l)]$, with $S(x_1, \dots, x_l) \overset{\text{def}}{=} \sum_{(y_1,\dots,y_l)<(x_1,\dots,x_l)} p(y_1, \dots, y_l)$,*
2. *encode $(x_1, \dots, x_l)$ with an element of the interval whose 2-adic representation length is $\lceil \log(p(x_1, \dots, x_l)) \rceil + 1$.*

This identifies the interval $\Rightarrow$ deduce $x_1 \dots x_l$ at the decoding step.

Advantage of arithmetic encoding

$$\textit{codelength}(x_1, \dots, x_l) = -\log p(x_1, \dots, x_l) + O(1).$$

$\Rightarrow$ at most $O(1)$ additional bits are wasted to encode l symbols and not $O(l)$ as before !

Very interesting in the previous example where 0's are generated with probability 2^{-10}. The average coding length becomes $h(2^{-10})l + O(1) \approx 0.011l$ instead of l for the Huffman code !

The major problem for implementing arithmetic coding

Need to compute $S(x_1, \ldots, x_l)$ *with a precision of* $\lceil \log p(x_1, \ldots, x_l) \rceil + 1$ *bits.*

$$\Downarrow$$

perform computations with arbitrary large precision.

The key idea which allows to work with packets of size l

Principle : efficient computation of $p(x_1, \dots, x_l)$ and $S(x_1, \dots, x_l)$.

- Computation of $p(x_1, \dots, x_l)$

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

- *iterative* computation of $S(x_1, \dots, x_l)$: *lexicographic order* on $\mathcal{X}^l$.

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

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

Proof

Let m be the smallest element in the alphabet

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

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

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

Proof (cont'd)

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

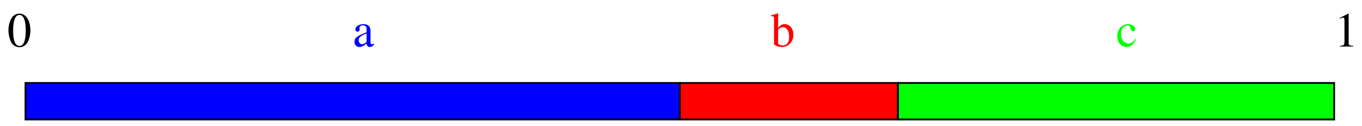
Proof (end)

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

fractals...

Example :

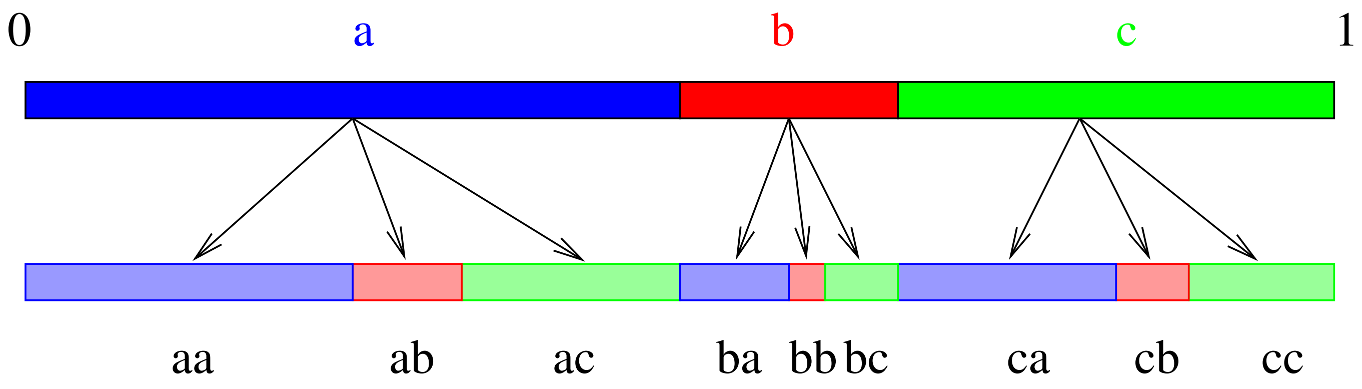
$$\mathbf{Prob}(a) = \frac{1}{2}; \mathbf{Prob}(b) = \frac{1}{6}, \mathbf{Prob}(c) = \frac{1}{3}.$$



fractals

Example :

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



coding

$$S(x_1,\dots,x_l) = \underbrace{S(x_1,\dots,x_{l-1})}_{\textit{beginning of the previous interval}} + \underbrace{p(x_1,\dots,x_{l-1})}_{\textit{length of previous interval}} S(x_l)$$

Therefore if the current interval is $[a,b[$*,and if* x_i *is read, then*

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

The width of the interval becomes $(b-a)p(x_i) = p(x_i)\dots p(x_{i-1})p(x_i) = p(x_1,\dots,x_i)$.

Zooming

Let $[a, b[$ be the current interval

- *If $b \leq 1/2$,* $\quad [a, b[\to [2a, 2b[$*, output ("0");*

- *If $a \geq 1/2$,* $\quad [a, b[\to [2a - 1, 2b - 1[$*, output ("1")*

- *If $1/4 \leq a < b \leq 3/4$,* $\quad [a, b[\to [2a - 1/2, 2b - 1/2[$*, no output (underflow expansion), but keep this in mind (with the help of a counter).*

Decoding

Let $r = \varphi(x_1, \ldots, x_l)$ *be the real number whose* 2*-adic representation encodes* $(x_1, \ldots, x_l) \in \mathcal{X}^l$*. The letters* $x_1, \ldots, x_l$ *are the only ones for which*

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

Procedure

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

Suppose that x_1 was found. We have

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

We have therefore to find out in which interval lies $r_1 = \frac{r-S(x_1)}{p(x_1)}$.