

Algorithmique et Analyse d'Algorithmes

L3 Info

Cours 9 : Codage et compression

Benjamin Wack



2024 – 2025

La dernière fois

- ▶ Arbre partiellement ordonné
- ▶ Structure de tas
- ▶ Application à la FAP
- ▶ Diviser pour Régner

Aujourd'hui

- ▶ Codage
- ▶ Entropie
- ▶ Algorithme de Huffman

Plan

Codage alphanébétique

- Notion de code et arbre

- Notion de code préfixe

Entropie

Algorithme de Huffman

- Fonctionnement

- Analyse

Codage

On dispose d'un alphabet de symboles $\mathcal{S} = \{s_1, \dots, s_k\}$ (caractères, lexèmes, notes de musique...).

Code

Un **code** (binaire) alphabétique C est une fonction :

$$\begin{aligned} C : \mathcal{S} &\longrightarrow \{0, 1\}^* \\ s_i &\longmapsto C(s_i) \text{ de longueur } l_i \end{aligned}$$

On code ensuite tout mot u de \mathcal{S}^* par

$$C(u_1 \dots u_n) = C(u_1) \dots C(u_n)$$

(peut être adapté pour un autre alphabet d'arrivée)

(Non-)Ambigüité du codage

Propriété désirable pour tout code

Le code C est **uniquement déchiffrable** si

$\forall c \in \{0, 1\}^*$, il existe **au maximum** un mot $u \in \mathcal{S}^*$ tel que $C(u) = c$

autrement dit :

- ▶ $C : \mathcal{S}^* \rightarrow \{0, 1\}^*$ est injective.
- ▶ Tout mot binaire ou bien admet un unique décodage, ou bien n'est pas un mot code.

Exemple de codage non ambigü

$\mathcal{S} = \{a, b\}$ et le code C tel que $C(a) = 01$, $C(b) = 10$.

Tout mot admet au plus un décodage.

Exemple de codage ambigü

$\mathcal{S} = \{x, y, z\}$ et le code C tel que $C(x) = 0$, $C(y) = 10$ et $C(z) = 01$.

Le mot 010 peut être décodé en xy ou en zx .

Exemple : le codage ASCII

- ▶ Alphabet à coder : ensemble de 128 caractères (256 pour ASCII étendu).
- ▶ Principe de codage : chaque symbole est codé sur 8 bits. (code de **longueur fixe**)

Propriété évidente

Tout code de longueur fixe est uniquement déchiffrable.

Efficacité d'un codage

- ▶ ASCII standard pourrait être codé sur 7 bits seulement : ce n'est pas un codage efficace.
- ▶ De même si on **sait que** tous les caractères ne sont pas utilisés, on peut définir un codage « maison » sur moins de bits.
- ▶ Tout alphabet de taille k admet un code de longueur fixe sur $\log_2 k$ bits.

Arbre de codage/décodage

On peut représenter tout code binaire par un arbre binaire :

- ▶ les branches gauches représentent 0, les branches droites 1 ;
- ▶ chaque symbole de \mathcal{S} correspond à un nœud de l'arbre (mais certains nœuds sont sans symbole) ;
- ▶ le code de s_i est donné par le chemin de la racine à son nœud.

Arbre du codage ASCII

- ▶ Tous les mots-codes sont de longueur 8, donc tous les chemins aussi.
- ▶ L'arbre est donc de hauteur 8.
- ▶ Tous les mots binaires de longueur 8 (en ASCII étendu) correspondent à un symbole donc l'arbre est **complet** et les symboles sont portés par des **feuilles**.

Exemple de code de longueur variable : le code Morse

lettre	code
A	.-
B	-...
C	-.-. .
D	-..

lettre	code
E	.
F	..-. .
G	--. .
H

Caractéristiques notables

- ▶ Longueur **variable**
- ▶ Les caractères les plus **fréquents** ont un code plus **court**.

Limitations

- ▶ Fréquence des caractères dépendante de la langue, du contenu...
- ▶ **Non uniquement déchiffrable** : $..-. = M(F) = M(EAE)$
- ▶ En pratique : pause dans la transmission entre deux codes
 \Rightarrow troisième symbole **implicite** dans l'alphabet d'arrivée

Code préfixe

Définition

Un code a la **propriété du préfixe** si aucun mot code n'est le préfixe d'un autre mot code.

Exemples

$C(x) = 0$, $C(y) = 10$ et $C(z) = 11$ est préfixe.

$C(a) = 01$ et $C(b) = 011$ n'est pas préfixe.

Code préfixe et arbre

Un code est préfixe si et seulement si dans son arbre, tous les symboles de \mathcal{S} sont portés par des feuilles.

Corollaire

Tout code de longueur fixe est préfixe.

Intérêt des codes préfixes

Propriété

Un code ayant la propriété du préfixe est **uniquement** déchiffrable (c'est-à-dire non ambigu).

C'est une condition suffisante mais pas nécessaire.

Démonstration

En effet, supposons qu'il y a des mots codes ambigus, et appelons c le mot code ambigu **le plus court** :

- ▶ on suit le début du chemin de c dans l'arbre de codage ;
- ▶ on arrive à un symbole s (sinon c n'est pas un mot code valide) ;
- ▶ pour décoder c on doit commencer par s (propriété du préfixe)
- ▶ si on est à la fin de c , il n'est pas ambigu sinon la fin de c est plus courte et ambiguë : absurde.

Mais quand peut-on construire un code préfixe et comment ?

Inégalité de Kraft

On considère un code C ayant la propriété du préfixe.

Soit $C(s_1), \dots, C(s_k)$ les mots-codes et l_1, \dots, l_k leurs longueurs.

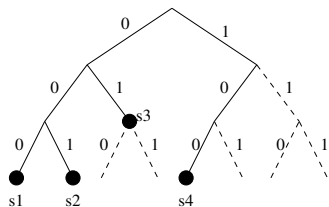
Inégalité de Kraft

Pour tout code ayant la propriété du préfixe

$$\sum_{i=1}^k \frac{1}{2^{l_i}} \leq 1 \quad (1)$$

Réciproquement, si (1) alors il existe un code avec la propriété du préfixe dont les mots-codes ont pour longueurs l_i .

Preuve de l'inégalité de Kraft



Symbole	Mot code	longueur
S_1	000	3
S_2	001	3
S_3	01	2
S_4	100	3

$$\sum_{i=1}^4 2^{-l_i} = \frac{1}{2^3} + \frac{1}{2^3} + \frac{1}{2^2} + \frac{1}{2^3} = \frac{5}{8} \leq 1.$$

Comme le code a la propriété du préfixe, dans son arbre de codage chaque mot code mène à une feuille.

On complète cet arbre.

Il est de hauteur $m = \max l_i$ la plus grande longueur des mots codes.

On compte les feuilles de profondeur m par mot code, puis globalement :

$$\sum_{i=1}^k 2^{m-l_i} \leq 2^m.$$

En divisant par 2^m on obtient l'inégalité de Kraft.

Qualité d'un code

Les codes préfixes assurent qu'ils sont décodables sans erreur (indispensable), mais certains sont-ils meilleurs que d'autres ?

La seule information dont on dispose est la fréquence dans le texte des k symboles à coder. On note ces fréquences $\{p_1, \dots, p_k\}$.

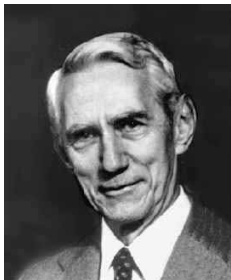
Longueur moyenne d'un code

$$L(C) = \sum_{i=1}^k p_i l_i$$

Longueur moyenne du code Morse (en anglais)

Symbole	Fréquence	Longueur du code	
A	0,082	2	Longueur moyenne $\simeq 2,54$
B	0,015	4	
C	0,028	4	
⋮	⋮	⋮	

Claude Shannon (1916-2001)



- ▶ Fondateur de la théorie de l'information
- ▶ A travaillé parallèlement au MIT et dans les laboratoires Bell
- ▶ Spécialisé dans les télécommunications : cryptographie (pendant la guerre), compression, capacité de transmission d'un canal...

En 1948, il associe à une séquence de lettres (ou de mots) son entropie (définition analogue à la définition en physique).

Cette entropie représente une **quantité d'information** contenue dans la séquence de lettres :

- ▶ une séquence redondante a une entropie faible
- ▶ une séquence aléatoire a une entropie importante

Théorème de Shannon

Théorème (Shannon 1948)

Pour le **meilleur** code préfixe sur un alphabet de fréquences $\{p_1, \dots, p_k\}$,

$$\mathcal{H}(p) \leq L(C) \leq \mathcal{H}(p) + 1.$$

où $\mathcal{H}(p)$ appelée *entropie* du système est définie par

$$\mathcal{H}(p) = - \sum_{i=1}^k p_i \log_2 p_i.$$

- ▶ Ce résultat lie la longueur moyenne du meilleur code aux fréquences **indépendamment du code choisi**.
- ▶ Borne absolue : on ne pourra pas compresser l'information au delà d'un certain facteur $\mathcal{H}(p)$.
- ▶ D'après Kraft, il est possible de construire un code préfixe en affectant à chaque symbole s_i un code de longueur $l_i = \lceil -\log_2 p_i \rceil$

Principe

On cherche un code binaire C :

- ▶ ayant la propriété du préfixe (facile à décoder)
- ▶ minimisant la longueur moyenne du code (efficace)

En pratique, on cherche à construire **l'arbre** de C .

Le principe

On construit une solution en partant des feuilles les plus profondes (codes les plus longs donc fréquences les plus faibles) puis en les combinant deux à deux on construit l'arbre.

Dans cet algorithme on utilise une structure de données :

- ▶ *file à priorité* F
- ▶ les éléments de la FàP sont des couples (nœud de l'arbre, fréquence)
- ▶ priorité d'un nœud = sa fréquence
- ▶ on extrait les éléments de poids **minimal d'abord**.

Algorithme de Huffman (1951)

entropie : $\mathcal{H}(p) \simeq 3.38$

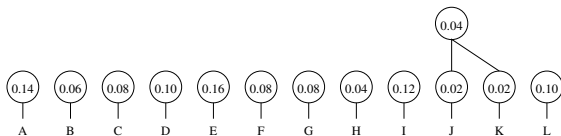
A	0.14
B	0.06
C	0.08
D	0.10
E	0.16
F	0.08
G	0.08
H	0.04
I	0.12
J	0.02
K	0.02
L	0.10



Algorithme de Huffman (1951)

entropie : $\mathcal{H}(p) \simeq 3.38$

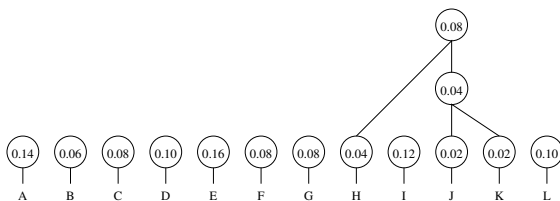
A	0.14
B	0.06
C	0.08
D	0.10
E	0.16
F	0.08
G	0.08
H	0.04
I	0.12
J	0.02
K	0.02
L	0.10



Algorithme de Huffman (1951)

entropie : $\mathcal{H}(p) \simeq 3.38$

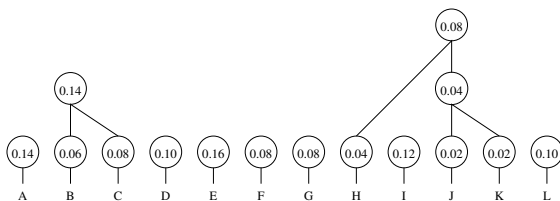
A	0.14
B	0.06
C	0.08
D	0.10
E	0.16
F	0.08
G	0.08
H	0.04
I	0.12
J	0.02
K	0.02
L	0.10



Algorithme de Huffman (1951)

entropie : $\mathcal{H}(p) \simeq 3.38$

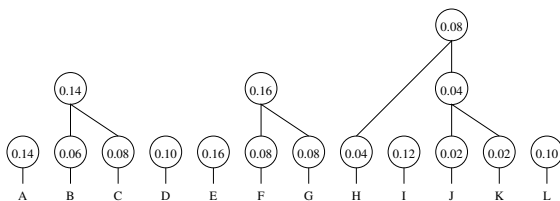
A	0.14
B	0.06
C	0.08
D	0.10
E	0.16
F	0.08
G	0.08
H	0.04
I	0.12
J	0.02
K	0.02
L	0.10



Algorithme de Huffman (1951)

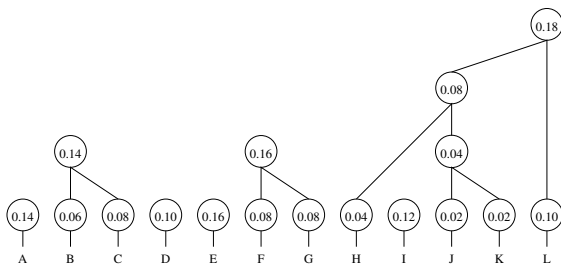
entropie : $\mathcal{H}(p) \simeq 3.38$

A	0.14
B	0.06
C	0.08
D	0.10
E	0.16
F	0.08
G	0.08
H	0.04
I	0.12
J	0.02
K	0.02
L	0.10



Algorithme de Huffman (1951)

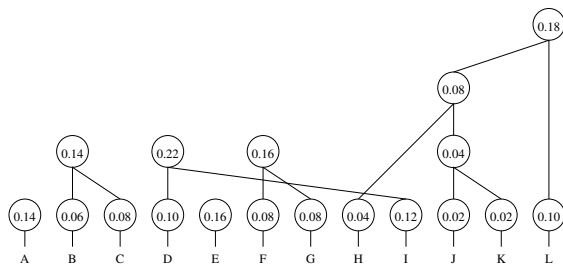
entropie : $\mathcal{H}(p) \simeq 3.38$



A	0.14
B	0.06
C	0.08
D	0.10
E	0.16
F	0.08
G	0.08
H	0.04
I	0.12
J	0.02
K	0.02
L	0.10

Algorithme de Huffman (1951)

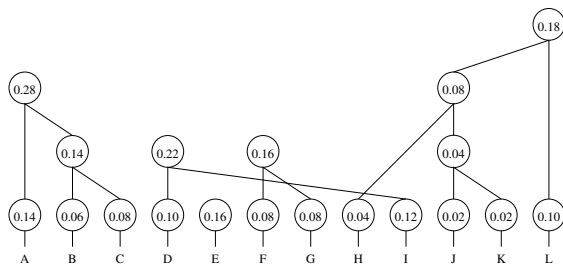
entropie : $\mathcal{H}(p) \simeq 3.38$



A	0.14
B	0.06
C	0.08
D	0.10
E	0.16
F	0.08
G	0.08
H	0.04
I	0.12
J	0.02
K	0.02
L	0.10

Algorithme de Huffman (1951)

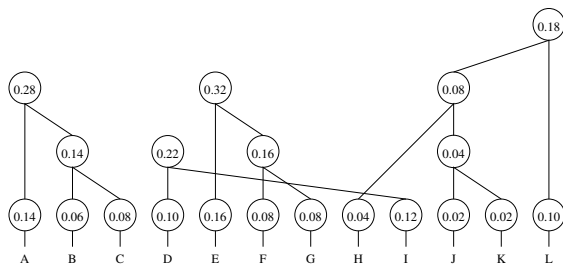
entropie : $\mathcal{H}(p) \simeq 3.38$



A	0.14
B	0.06
C	0.08
D	0.10
E	0.16
F	0.08
G	0.08
H	0.04
I	0.12
J	0.02
K	0.02
L	0.10

Algorithme de Huffman (1951)

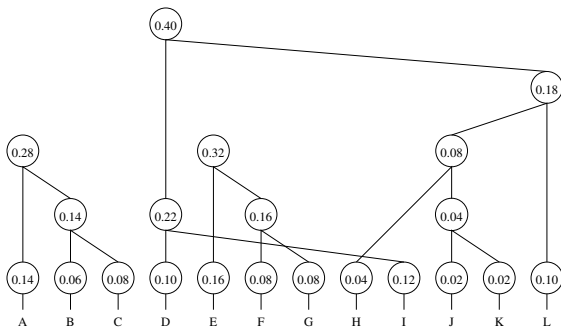
entropie : $\mathcal{H}(p) \simeq 3.38$



A	0.14
B	0.06
C	0.08
D	0.10
E	0.16
F	0.08
G	0.08
H	0.04
I	0.12
J	0.02
K	0.02
L	0.10

Algorithme de Huffman (1951)

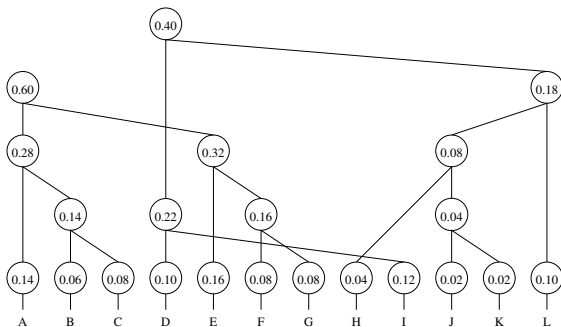
entropie : $\mathcal{H}(p) \simeq 3.38$



A	0.14
B	0.06
C	0.08
D	0.10
E	0.16
F	0.08
G	0.08
H	0.04
I	0.12
J	0.02
K	0.02
L	0.10

Algorithme de Huffman (1951)

entropie : $\mathcal{H}(p) \simeq 3.38$



A 0.14

B 0.06

C 0.08

D 0.10

E 0.16

F 0.08

G 0.08

H 0.04

I 0.12

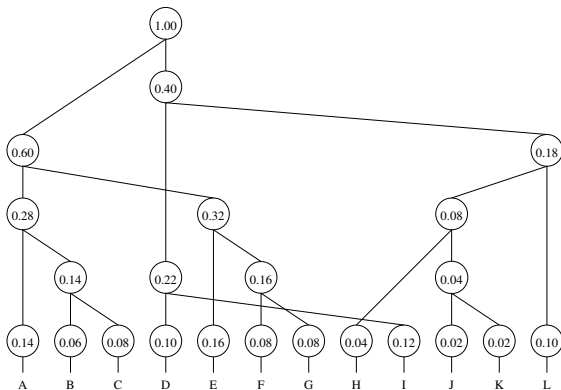
J 0.02

K 0.02

L 0.10

Algorithme de Huffman (1951)

entropie : $\mathcal{H}(p) \simeq 3.38$



A 0.14

B 0.06

C 0.08

D 0.10

E 0.16

F 0.08

G 0.08

H 0.04

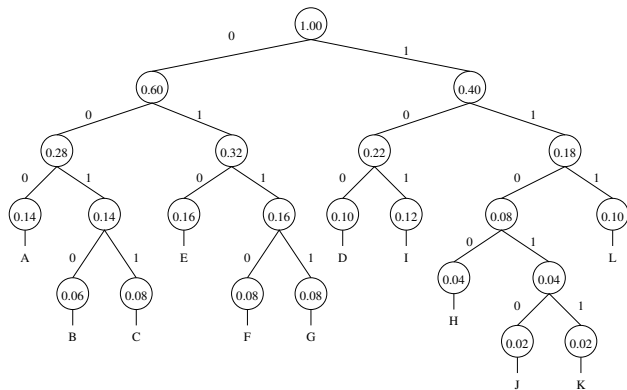
I 0.12

J 0.02

K 0.02

L 0.10

Algorithme de Huffman



A	0.14	000
B	0.06	0010
C	0.08	0011
D	0.10	100
E	0.16	010
F	0.08	0110
G	0.08	0111
H	0.04	1100
I	0.12	101
J	0.02	11010
K	0.02	11011
L	0.10	111

- ▶ Codage optimal : longueur moyenne $L(C) = 3.42$
(rappel de l'entropie $\mathcal{H}(p) \simeq 3.38$)
- ▶ Profondeur de chaque symbole $s_i \simeq -\log_2 p(s_i)$

Algorithme de Huffman : Implantation

ALGORITHME_HUFFMAN

Données : Un ensemble S de k symboles de fréquences p_1, \dots, p_k

Résultat : Un arbre optimal de codage (son nœud racine)

Indice i

Nœud x, y, z

FàP $F = F\grave{a}PVide()$ // File à priorité de nœuds pondérés

pour $s \in S$

$z = \text{Noeud}(\text{ArbreVide}, \text{symbole}=s, \text{poids}=p(s), \text{ArbreVide})$
 Insérer ($F, z, p(s)$)

pour $i = 1$ à $K - 1$

$x = \text{Extraire}(F)$
 $y = \text{Extraire}(F)$
 $z = \text{Noeud}(x, \text{symbole}=\dots, \text{poids} = \text{poids}(x) + \text{poids}(y), y)$
 Insérer ($F, z, \text{poids}(z)$)

renvoyer Extraire (F)

Algorithme de Huffman : Remarques

Non-unicité

Pour un même alphabet, on peut produire des arbres très différents pour plusieurs raisons :

- ▶ Si plusieurs éléments de fréquence minimale, on peut choisir n'importe lequel
- ▶ Pour chaque nœud, le choix de sa gauche et de sa droite est libre.

Conséquences

Pour décoder, il faut :

- ▶ Fournir explicitement le code retenu
- ▶ Ou au moins décider une méthode non ambiguë de choix dans les deux situations ci-dessus.

Codage et décodage

Codage

On se contente de parcourir le texte à coder, et de remplacer chaque symbole par son mot code : **la table des codes suffit.**

Décodage

Propriété du préfixe : dès qu'un mot code est **reconnu** on le décode.

- ▶ En théorie la table suffit.
- ▶ Pour être efficace on utilise l'arbre :
 - ▶ on suit le chemin désigné par la suite de bits initiale du mot binaire ;
 - ▶ tant qu'on n'est pas arrivé à une feuille on continue ;
 - ▶ si on arrive à une feuille on a son symbole et on repart de la racine.

Dans le cadre de la compression, le fichier compressé doit donc fournir :

- ▶ les données codées (gain de taille)
- ▶ mais aussi l'arbre ou la table (poids supplémentaire ou *overhead*)

Algorithme de Huffman : Preuve

Optimalité

L'algorithme de Huffman produit un code ayant la propriété du préfixe de longueur moyenne optimale.

Lemme : fréquences faibles

Soit C un alphabet de k lettres.

Soient x et y deux symboles de plus petite fréquence.

Alors il existe un codage préfixé optimal pour C tel que les mots codes de x et de y ne diffèrent que par le dernier bit.

Idee : prendre un arbre optimal et le transformer de manière à vérifier la propriété.

(Il existe deux feuilles sœurs parmi les plus profondes de l'arbre, et on peut échanger leurs symboles avec x et y sans augmenter la longueur moyenne du code.)

Algorithme de Huffman : Preuve

Lemme : propagation de l'optimalité

Soit T un arbre de codage optimal (terminé) de C .

Alors :

- ▶ la fusion z de 2 feuilles sœurs x et y
- ▶ affectée de la somme des fréquences des feuilles $f(z) = f(x) + f(y)$

produit un arbre optimal pour l'alphabet C' dans lequel tous les symboles x et y ont été remplacés par z .

Idee : démonstration par l'absurde.

Algorithme de Huffman : Preuve

Invariant de la boucle :

La file à priorité contient une forêt incluse dans un arbre de codage optimal de l'alphabet C .

(trivialement vérifié en début de l'algorithme)

Preuve partielle : Supposons qu'en début d'une itération, il existe un arbre optimal contenant la forêt incluse dans la file à priorité.

Soient x et y les nœuds extraits de la FàP :

- ▶ D'après le lemme 2 (propagation de l'optimalité), on peut faire comme si x et y étaient des feuilles codant chacune un caractère (de mêmes fréquences que x et y respectivement).
- ▶ D'après le lemme 1 (fréquences faibles) on peut modifier l'arbre optimal pour que x et y soient 2 feuilles sœurs.
- ▶ Donc en créant un nœud dont les filles sont x et y la forêt obtenue reste une sous-forêt d'un arbre optimal.

Algorithme de Huffman : Complexité

Complexité

- ▶ Chaque itération construit un nœud interne d'un arbre à k feuilles.
- ▶ L'algorithme nécessite donc $k - 1$ itérations.
- ▶ Si la file à priorité est implantée dans un tas (*cf* cours 8), les coûts d'insertion et d'extraction sont en $\mathcal{O}(\log k)$.

D'où un **algorithme en $\mathcal{O}(k \log k)$** pour un alphabet de k symboles.

Terminaison

On a prouvé la terminaison par la même occasion, car le nombre d'itérations est toujours de $k - 1$.

(Le nombre de nœuds dans la FàP est un variant de boucle.)

En résumé

Aujourd'hui

- ▶ Le **codage** consiste à représenter une information à l'aide de **mots** dans un autre alphabet (usuellement binaire).
- ▶ Un **bon code** doit pouvoir être **décodé** mais aussi produire des messages **de longueur moyenne faible**.
- ▶ L'**entropie** donne une **borne inférieure** pour cette longueur moyenne pour une **répartition donnée** des symboles.
- ▶ L'**algorithme de Huffman** donne une méthode pour **construire** un code réalisant cette borne inférieure.

La prochaine fois

- ▶ Problèmes d'optimisation
- ▶ Algorithmes gloutons
- ▶ Algorithmique de graphes et problèmes de coloration