
© UC SANTA CRUZ

**Pour réduire la taille d'un texte dans un fichier informatique, rien de tel que de dessiner un arbre. On retrouve ainsi plus vite les lettres qui reviennent le plus souvent, et on économise de la place.**

# Des arbres à compresser

**Hervé Lehning,**
**professeur de mathématiques spéciales au lycée Janson-de-Sailly.**
herve.lehning@prepas.org

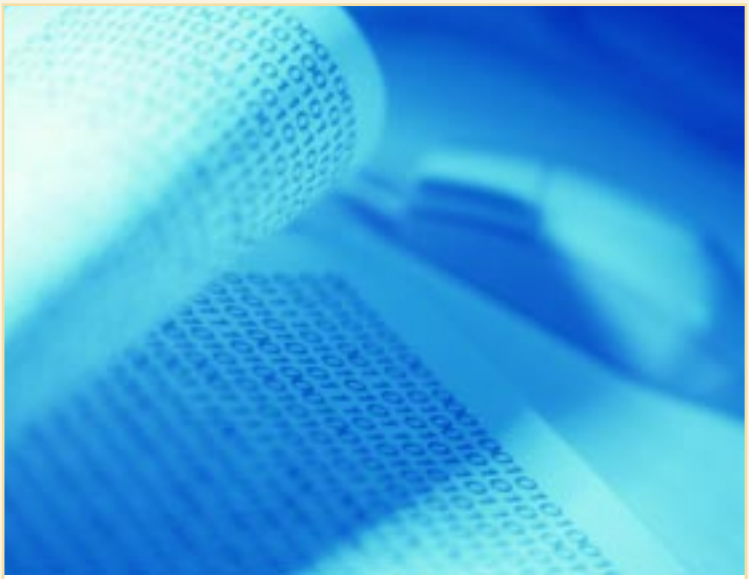
*« Par ma foi ! il y a plus de quarante ans que je dis de la prose sans que j'en susse rien. »* Dans cette phrase extraite du *Bourgeois gentilhomme,* monsieur Jourdain utilise, sans le savoir non plus, 10 « e », 8 « a » et seulement 3 « q » et 2 « j ». L'ordinateur ne le sait pas plus lorsqu'il lit le fichier numérique permettant d'afficher ce texte. Et pourtant, cette redondance peut s'avérer utile. Comment ?

Avant de se plonger dans l'analyse des lettres, rappelons que, quelle que soit leur nature (texte, image ou son), les fichiers informatiques sont des suites d'octets, c'est-à-dire des groupes de huit bits (0 ou 1). Compresser un fichier signifie le transformer en un autre fichier de plus petite taille. La compression se fait au moyen d'algorithmes. Elle est dite « sans perte » s'il est toujours possible de reconstituer l'original. Que l'on compresse une chanson ou un film, le travail de l'algorithme est le même : il s'agit de réduire la taille des suites d'octets. Un acte devenu banal dans l'usage grand public de l'informatique.

## Huit bits = 256 valeurs

Un groupe de huit bits peut prendre 28 = 256 valeurs différentes (toutes les combinaisons de 0 et de 1 comprises entre 00000000 à 11111111) et permet donc de coder 256 caractères suivant, par exemple, le code ASCII (American Standard Code for Information Interchange). Celui-ci faisait initialement appel à sept bits ; il a été étendu à huit bits pour inclure les caractères accentués (qui n'étaient pas prévus à l'origine car le code avait été conçu pour l'anglais).

Comme nous l'avons évoqué, tous les caractères ne sont pas utilisés avec la même fréquence. D'où l'idée de coder les plus fréquents avec moins de bits que les autres. Pour ses premiers Macintosh, la société Apple a ainsi imaginé de crypter les quinze caractères les plus répétés – en français, ce sont les voyelles e, a, i, etc. ainsi que les signes de ponctuation – sur quatre bits. Une seizième configuration des quatre bits (par exemple 1111) est réservée pour signifier : « il ne s'agit pas de l'un des 15 caractères les plus courants. » Le caractère en question est alors codé avec huit autres bits – donc douze en



**LXXXXXXXXXX XXXXXXXE à transmettre est compSequis et iure molore magnim nit, quat, velit del ute tinim nostie ming ex el digna feum doleniamet aute dolore consequam ent lobor sit in xxxxxxxxven 200 s.** © MARK WRAGG/IMAGESTATE/EYEDEA.FR

tout. Douze bits au lieu de huit pour les caractères les moins fréquents, quatre au lieu de huit pour les plus fréquents : si ces derniers comptent pour 80 % de l'ensemble des caractères, alors d'un côté on perd 10 % (la moitié de 20 %) et de l'autre, on gagne 40 % (la moitié de 80 %). Le gain moyen est finalement de 30 %.

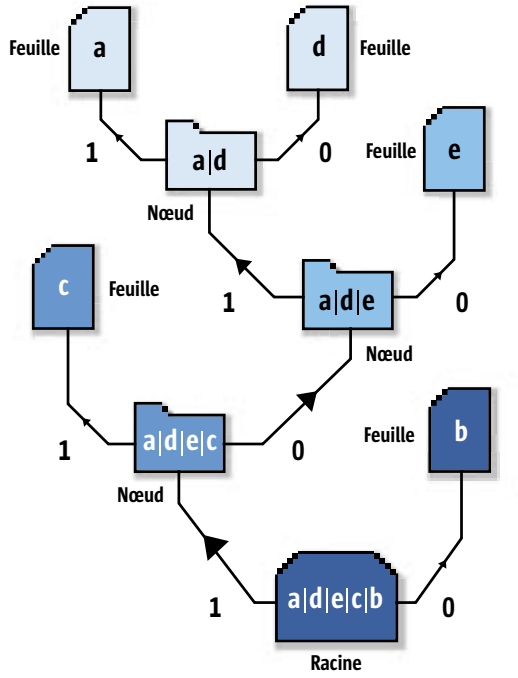
## Nœuds père et fils

Peut-on améliorer ce résultat ? Oui, à condition d'accepter que les méthodes soient plus longues à appliquer. L'une des solutions les plus célèbres est due à l'informaticien américain David Huffman. En 1952, alors qu'il était étudiant au Massachusetts Institute of Technology, il eut l'idée de faire varier davantage la longueur des codes en écrivant le caractère le plus fréquent avec un bit, le suivant avec deux et ainsi de suite [1]. Il utilisa des codes dont aucun n'est le préfixe d'un autre, comme 0, 101, 111, 100 et 1101, afin de pouvoir les distinguer même lorsqu'ils sont écrits les uns à la suite des autres. Par exemple, avec le codage ci-dessus, la suite 11111010100 se découpe en 111, 1101, 0, 100.

Comment créer de tels codes ? Imaginons un arbre dont les feuilles seraient des lettres. Cet arbre est un « arbre binaire ». Il s'agit d'une structure de données composée de nœuds : chaque nœud a un père (sauf le premier, que l'on appelle la racine) et deux fils (sauf les derniers, que l'on appelle les feuilles). Le tout se présente comme un arbre généalogique [fig. 1]. En partant de la racine, on atteint chaque lettre en empruntant successivement les branches dans une direction ou dans une autre. On code le trajet par une suite de bits en notant, à chaque étape, 0 ou 1, selon que l'on emprunte la branche de gauche ou celle de droite. Enfin, on attribue à chaque lettre, le code du trajet qui y mène.

Cette façon de procéder nous assure qu'aucun code n'est le préfixe d'un autre, car un chemin vers une feuille ne peut passer par une autre feuille. La longueur d'un code correspond à la « hauteur » de la lettre dans l'arbre. Les feuilles les plus hautes représentent donc les caractères les moins fréquents.

## Fig.1 L'algorithme de Huffman




**LE TEXTE à transmettre est composé des lettres a, b, c, d et e. Leurs fréquences d'apparition sont respectivement égales à 5 %, 50 %, 20 %, 10 % et 15 %. L'arbre de Huffman se construit en partant du bas et en remontant. Il s'utilise ensuite dans le sens contraire. Les codes des lettres correspondent au trajet pour les atteindre depuis la racine, soit 0 pour *b*, 11 pour *c*, 100 pour *e*, 1010 pour *d* et 1011 pour *a*.** © INFOGRAPHIE BRUNO BOURGEOIS

## Greffe de lettres

Concrètement, comment procède-t-on pour construire un tel arbre ? Tout d'abord, il faut classer les lettres suivant leur fréquence d'apparition dans le texte. On retire de cette liste les deux qui se révèlent être les moins fréquentes, disons y et z. On greffe ces deux lettres à une racine : elles deviennent respectivement le « fils gauche » et le « fils droit » de ce nœud – la racine dans ce premier cas – et l'on obtient un petit arbre noté y|z. On additionne maintenant la fréquence de y et celle de z. La nouvelle valeur est celle de l'arbre y|z : elle peut être intégrée dans notre classement, l'arbre prenant la place des deux lettres et la racine devenant un nœud. Il ne reste plus qu'à répéter l'opération autant de fois que nécessaire pour obtenir un arbre unique. C'est l'arbre de Huffman du texte.

## Limites du codage par octets

Ce codage permet d'obtenir une compression d'environ 40 % en moyenne : personne n'a réussi à faire mieux. Lorsque l'on évoque des compressions de 80 % pour du texte, c'est que l'on a abandonné le codage par octets. En effet, en s'affranchissant de cette contrainte, on peut apporter des améliorations fondées, par exemple, sur la création d'un dictionnaire des mots rencontrés : avec un tel dictionnaire, on peut remplacer un mot par son « adresse » [2]. Cette manière de procéder est surtout efficace pour les textes car, pour les autres types de fichiers, on rencontre moins de longs mots répétitifs. De plus, ces algorithmes améliorés demandent des temps d'exécution importants. Pour un codage et un décodage rapide, l'algorithme de Huffman garde l'avantage.

La démarche décrite ici est une stratégie classique d'optimisation. L'idée sous-jacente consiste à dire que l'on peut arriver à une solution optimale en effectuant un choix optimal à chaque étape. Un tel choix est qualifié de « glouton » car il évoque un goinfre se jetant toujours sur le plus gros morceau sans réfléchir à la suite. En procédant ainsi à chaque étape, l'algorithme peut conduire à une solution optimale – c'est le cas avec la méthode de Huffman –, mais ce n'est pas toujours le cas.

Cet algorithme est très différent de ceux couramment utilisés aujourd'hui pour la compression d'images ou de sons : ceux-ci, jouant sur les faiblesses des sens humains, sont destructifs. C'est le cas des ondelettes, utilisées pour le standard Jpeg2000 [3]. Cependant, on utilise l'algorithme de Huffman après une première compression avec perte pour réduire encore la taille du fichier. ■

[1] D. Huffman, *Proceedings of the I.R.E., 40,* 1098, 1952.

[2] A. Lempel et J. Ziv, *IEEE Transactions on Information Theory, 24,* 530, 1978.

[3] M. Nowak et Y. Meyer, « La surprenante ascension des ondelettes », *La Recherche,* février 2005, p. 56.

## POUR EN SAVOIR PLUS

Xavier Marsault, *Compression et cryptage des données multimédia,* Hermès, 1995.