République Algérienne Démocratique et Populaire
Ministère de l'Enseignement Supérieur et de la Recherche Scientifique

Université de Ghardaïa

Faculté des Sciences et Technologie
Département des Mathématiques et Informatique

Projet de fin d'étude présenté en vue de l'obtention du diplôme de

LICENCE

Domaine : Mathématiques et Informatique

Spécialité : Informatique

THEME:

Compression de données,

Méthodes et Applications

PAR :
Souad BAZEMLAL
Zineb ABAZA

Jury:

Mr: Slimane BELLAOUAR	Maitre Assistant A Univ. Ghardaia	**Encadreur**
Mr: Abdelkader Ouled MEHREZ	Maitre Assistant B Univ. Ghardaia	**Examinateur**

ANNEE UNIVERSITAIRE: 2013/2014

République Algérienne Démocratique et Populaire
Ministère de l'Enseignement Supérieur et de la Recherche Scientifique

Université de Ghardaïa

Faculté des Sciences et Technologie
Département des Mathématiques et Informatique

Projet de fin d'étude présenté en vue de l'obtention du diplôme de

LICENCE

Compression de données,

Méthodes et Applications

PAR :

Souad BAZEMLAL
Zineb ABAZA

Dirigé par :

Slimane BELLAOUAR

Remerciements

Nous tenons à remercier tout d'abord Dieu tout puissant de nous avoir donné santé, courage et volonté d'accomplir ce modeste travail.

Nous présentons également nos sincères remerciements

A monsieur Slimane BELLAOUR d'avoir accepté de nous encadrer, ainsi pour la richesse et la qualité de son enseignement et les efforts consacrés pour une bonne formation.

A monsieur Bachir BENYAMMI, nous tenons à exprimer nos estimes et nos gratitudes pour son aide, ses encouragements et sa présence.

A nos professeurs Youcef MAHJOUB, Slimane OULAD NAOUI, Abdelkader OULED MAHREZ, Toufik GHARIB et Djelloul ZIADI pour l'enseignement qu'ils nous ont transmis.

A Ayman et tous ceux qui ont contribué pour mettre en place ce travail.

Zineb ABAZA et Souad BAZEMLAL

Dédicaces

A Mes chers parents avec tous mes sentiments de respect, de gratitude et de reconnaissance pour tous les sacrifices déployés pour m'élever et assurer mon éducation dans les meilleures conditions, que dieu leur procure bonne santé et longue vie.

A Ma grande mère qui nous a toujours encouragées avec ses prières, que dieu la protège.

A Mes sœurs Fatiha, Meriem et Rokia.

A Mes frères Salah, Abdennour et le petit Nabil.

A Ma chère copine et binôme Souad

A et Toutes mes amies et connaissances.

Je dédie ce travaille.

Zineb ABAZA

Dédicaces

J'ai le plaisir de dédier ce mémoire à :

Mes parents qui œuvrent à ma réussite par leur amour, leur soutien et les valeurs qu'ils m'ont transmises.

A mes grands parents.

A Daoud, Fella et Fatiha pour leurs affection et encouragements.

A l'adorable Yasmine pour qui j'espère que la vie lui réserve le meilleur.

A mes tentes et mes oncles.

A mes collègues de promo, Zineb, Habiba et Nassima.

A Hadjer et tous mes amis.

Souad BAZEMLAL

Table des matières

Liste des figures

Liste des tableaux

Liste des acronymes

AAC : Advanced Audio Coding

ASCII : American Standard Code for Information Interchange

AIFF : Audio Interchange File Format

ADPCM : Adaptatif différential Pulse Code Modulation

API : Application Program Interface

CD : Compact Disk

DWT : Discrete Wavelet Transform

DCT : Discrete cosine Transform

DVD : Digital Vidéo Disk

FDA : Food and Drug Administration

FIF : Fractal Image Format

FLAC : Free Lossless Audio Codec

GIF : Graphics Interchange Format

GSM : Global System Mobile

IFS : Iterated Function Systems

ISO : International Organization for Standardisation

IEEE : Institue of Electrical and Electronic Enginneers

JPEG : Joint Photographic Experts Group

lZW : Lempel Ziv Welsh

MIC : Modulation par Implusion codée

MJPEG : Motion Joint Photographic Expert Group

MPEG : Moving Picture Experts Group

MP3 : MPEG-1/2 Audio Layer 3

PNG : Portable Network Graphics

PDF : Portable Document File

RGB : Red Green Blue

RLE : Run Length Encoding

TIFF : Tagged Image File Format

UNICODE : Universal Code

UTF : Universal Transfer Format

VESA : Video Electronics Standard Association

WMA : Windows Media Audio

WAV : Windows Waveform sound

Résumé

La compression de données est un ensemble de méthodes visant à réduire la taille de l'information. L'augmentation spectaculaire d'informations stockées et échangées dans le monde fait de la compression un outil indispensable.

Dans ce travail, nous avons présenté six méthodes de compression largement utilisées notamment dans le multimédia. Trois d'entre elles, RLE (Run Length Encoding), Huffman et LZW (Lempel Ziv Welsh), permettent de réduire la taille de l'information sans perte de données, les trois autres, DCT (Discrete Cosine Transform), Ondelette et fractales, tolèrent la perte de quelques données lors de la compression. Pour mettre en valeur la technique de compression de données nous avons présenté quelques domaines d'application.

Abstract

Data compression is a set of methods used to reduce the size of the information. The large increase of information stored and exchanged in the world makes compression as essential tool.

In this work, we presented six compression methods widely used particularly in multimedia. Three of them, RLE (Run Length Encoding), Huffman and LZW (Lempel Ziv Welsh), reduce the size of information without data loss, the three others, DCT (Discrete Cosine Transform), wavelet and fractal, tolerate some data loss during compression. To enhance the compression technique we presented some fields of application.

Introduction

L'avancée de la technologie et l'exigence des utilisateurs conduit à des quantités d'informations importantes, ce qui a amené à l'apparition du phénomène ''Big Data'' vers les années 2000, par conséquent, l'optimisation des voix de communication et les capacités de stockage est devenu un enjeu majeure en économie et un sujet d'actualité et d'étude.

La compression de données consiste à écrire l'information de manière compacte sans perdre les données ou en perdant les moins signifiantes.

Les méthodes de compression sont nombreuses, chacune exploite les propriétés des données à compresser, par exemple, la redondance dans un texte ou l'insensibilité de l'humain sur quelques informations dans le son et l'image.

Dans le premier chapitre nous introduisons quelques techniques de stockage de données dans la machine afin d'avoir une idée sur la représentation de l'information avant de procéder à la compression. Ensuite, nous présentons des définitions concernant la compression et ses classifications.

Dans le second chapitre, nous abordons les méthodes de compression sans perte, où les données peuvent être récupérées entièrement après la décompression. A titre d'exemple, nous décrivons les méthodes de compression RLE, la méthode Huffman et en fin la méthode LZW.

Dans le troisième chapitre, nous présentons trois méthodes de compression avec perte, où les données moins importantes dans le son ou l'image sont éliminées, à savoir, la méthode de compression par transformée en cosinus discrète (DCT), la compression par transformée en ondelettes et en fin la compression par fractales.

Le quatrième chapitre se focalise sur les domaines d'application des méthodes de compression abordées ainsi que les logiciels utilisés dans la compression.

Le mémoire est finalisé par une conclusion générale suivie par des annexes qui montrent l'implémentation de quelques algorithmes de compression en langage de programmation JAVA.

Chapitre 1

Généralités et définitions

Introduction

Le codage est le fait de représenter les données d'une façon précise, il permet de passer d'une représentation à une autre. En exploitant la redondance et l'insensibilité de l'œil et l'oreille humaine sur quelques données il est possible de réduire la taille de ces données, c'est le principe de la compression de données.

La compression est un outil appliqué à l'information et à la donnée, elle est utilisée dans les systèmes d'informations où les ressources (matérielles, logiciels, …) sont organisées, regroupées, classifiées et diffusées dans un environnement donné.

Codage de l'information

Histoire du codage

Depuis les temps anciens le codage des informations a permis à l'Homme de se communiquer via des dessins en premier et par des alphabets par la suite plus simple à utiliser et qui offrent de multiple combinaisons pour une plus grande richesse de l'expression.

Vers la fin des années 30 du siècle dernier, Claude Shannon démontre qu'à l'aide des interrupteurs on peut effectuer des opérations logiques on associant 1 à vrai (cas d'interrupteur fermé) et 0 à faux (cas d'interrupteur ouvert), ainsi se développe le langage machine qui permet de communiquer avec ou via la machine.

De nos jours, quelle que soit l'information sur la machine, elle est codée sous forme de nombre à base 2, ce qui est le langage binaire.

Le terme bit signifie Binary Digit qui est le 0 ou le 1 sous forme binaire, qui est la plus petite représentation dans la machine.

Le terme octet (Byte en anglais) signifie un ensemble de 8 bits. Il permet de représenter une lettre, un chiffre ou un caractère. A titre d'exemple '00110111' est la représentation binaire du chiffre '7', '01100001' signifié la lettre 'a' et '01000000' représente le caractère '@'.

Unités du codage

Les unités de mesure suivantes sont utilisées en informatique pour quantifier la taille de la mémoire d'un ordinateur, l'espace utilisable sur un disque dur, la taille d'un fichier, d'un répertoire ou autre.

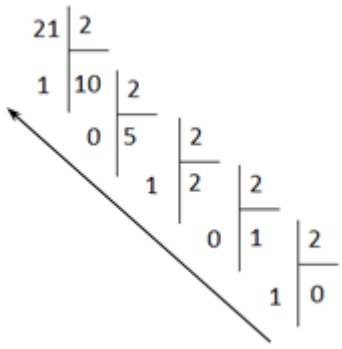
Tableau 1: Les unités de mesure

Unité	Valeur (octet)
Kilooctet	10^3
Mégaoctet	10^6
Gigaoctet	10^9
Téraoctet	10^{12}
Pétaoctet	10^{15}

Codage des nombres

Un nombre peut être un entier signé ou non signé, ou un réel
Pour coder un entier non signé il suffit de le convertir en binaire.

Exemple : 21 en binaire donne ''10101'' obtenu à l'aide de devisions successive de 21 par 2, le nombre binaire est le reste de divisions comme expliqué ci dessous :

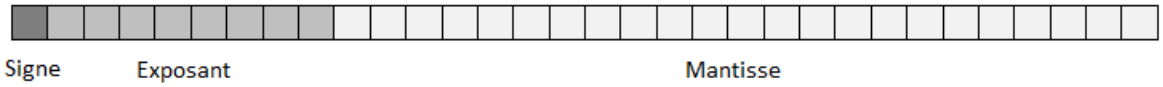


Un entier signé est codé en utilisant son complément à 2.

Le principe est d'inverser les bits de la valeur absolue puis d'additionner le résultat à 1.

Exemple : le complément à 2 du nombre 21 est l'inverse de sa valeur absolue (sur 8 bits par exemple) qui donne ''1110 1010'' puis en additionnant à 1 ça donne ''1110 1011''

Pour un réel, une norme existe, la norme IEEE 754.
Cette norme permet de représenter un réel sur 32 bits de la manière suivante :



- le signe est représenté par un seul bit, le bit de poids fort (le plus à gauche)

- l'exposant est codé sur les 8 bits consécutifs au signe
- la mantisse (les bits situés après la virgule) sur les 23 bits restants

Codage des caractères

Afin de coder les caractères, elle existe la norme ASCII qui permet d'associer à chaque caractère un nombre binaire sur 7 bits, Cependant, cette norme ne suffit pas à coder l'ensemble des lettres latines, le code ASCII étendu apparait afin d'introduire les lettres accentuées et d'introduire d'autres caractères, les caractères sont ainsi codés sur 8 bits.

Voici un extrait de la table ASCII :

Tableau 2 : Extrait de la table ASCII

Code	Caractère	Code	Caractère
00100000		00111110	>
00100001	!	00111111	?
00100010	"	01000000	@
00100011	#	01000001	A
00100100	$	01000010	B
00100101	%	01000011	C
00100110	&	01000100	D
00100111	'	01000101	E
00101000	(	01000110	F
00101001	)	01000111	G
00101010	*	01001000	H
00101011	+	01001001	I
00101100	,	01001010	J
00101101	-	01001011	K
00101110	.	01001100	L
00101111	/	01001101	M
00110000	0	01001110	N
00110001	1	01001111	O
00110010	2	01010000	P
00110011	3	01010001	Q
00110100	4	01010010	R
00110101	5	01010011	S
00110110	6	01010100	T
00110111	7	01010101	U
00111000	8	01010110	V
00111001	9	01010111	W
00111010	:	01011000	X
00111011	;	01011001	Y
00111100	<	01011010	Z
00111101	=		

Il existe d'autres normes que l'ASCII comme UNICODE sur 16 bits (UTF-16) qui contient une dizaine de milliers de code, mais les 128 premiers restent compatibles avec ASCII, ou encore UTF-8, UTF-32 et bien d'autres.

Codage de l'image

Il existe deux modes de codage d'une image : matricielle et vectorielle.

Une image matricielle (appelée aussi bitmap) est une grille composée par des petits carrés qu'on les appelle pixels, l'image est caractérisée par le nombre de carrés en hauteur est en longueur de la grille.

Chacun de ces pixels a une couleur représentée en binaire :

- 1 bit pour chaque pixel pour une image en noir et blanc.
- 1 octet pour chaque pixel pour une image avec 256 couleurs.
- 3 octet pour chaque pixel pour une image en couleur vrai (16 millions de couleurs).

Ce type de codage offre l'avantage d'être facile à représenter et à lire.

Une image vectorielle est un ensemble d'objets géométriques (lignes, cercles, textes, courbes, …) définis par leurs attributs (coordonnées, couleur, épaisseur de trait, …)

Ce type d'image est utilisé dans les dessins industriels, Systèmes d'information géographiques, … . Son avantage est qu'il occupe moins d'espace qu'une image bitmap et qu'il peut facilement être redimensionné .Ce format est utilisé pour les fichiers PDF.

Exemple :

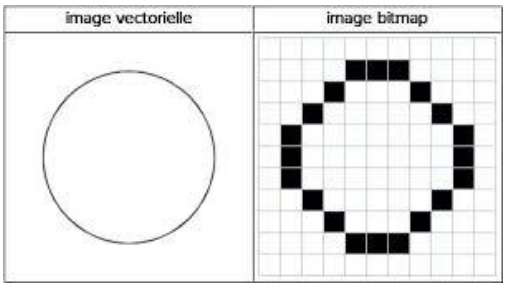


Figure 1: Différence entre image Matricielle et Vectorielle

Codage du son

Le son se présente par un signal analogique qui peut être numérisé en deux étapes : l'échantillonnage, qui consiste à découper le temps en éléments élémentaires, et la mesure de l'amplitude qui consiste à coder le niveau du signal.

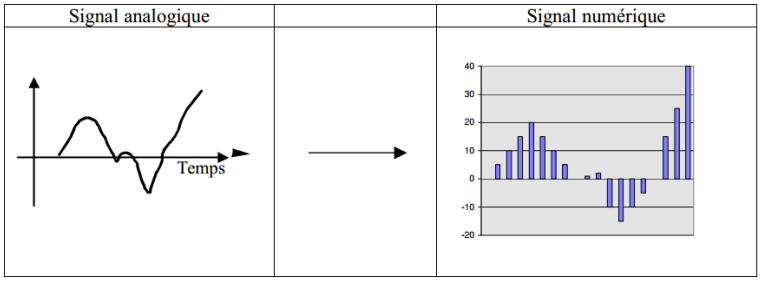


Figure 2: Passage du signal analogique au signal numérique

Dans le codage MIC (Modulation par Impulsion Codée), le signal est mesuré toute les 125 ms donc 8000 fois par seconde, le niveau d'amplitude est ensuite codé sur 8 bits. Le signal a donc un débit de 8*8000 = 64 Kbit/s.

Ce codage est utilisé dans le format populaire WAV (WAVE) et AIFF (Audio Interchange File Format).

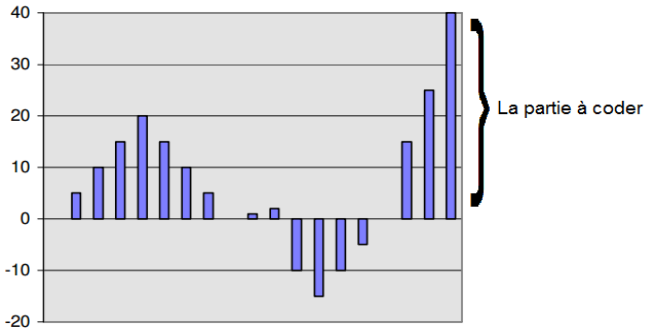


Figure 3: Codage MIC

Le codage ADPCM (Adaptatif Differential Pulse Code Modulation) évalue la différence entre le niveau du signal d'un échantillon avec celui de l'échantillon précédant.

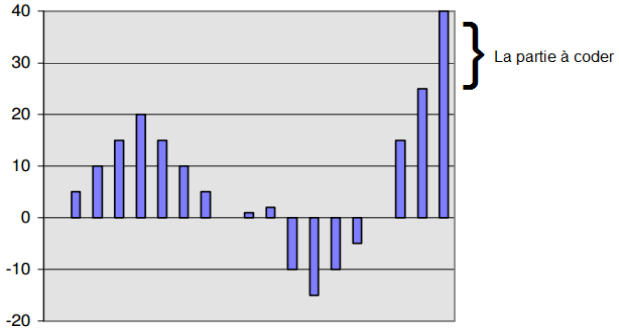


Figure 4: Codage ADPCM

Cette méthode utilise moins d'espace mémoire que le codage MIC car la différence entre les niveaux du signal entre deux échantillon successifs est moins grande que le niveau de signal lui-même.

Codage de vidéo

Une vidéo est une succession d'images. L'œil humaine a comme caractéristique la possibilité de distinguer 20 images par seconde, par suite si l'on affiche plus de 20 images par seconde, l'œil considère la séquence comme animation.

Définitions

Petit historique

Avec l'évolution technologique on est amené à manipuler des donnés de plus en plus importantes, à les stocker et les transférer. Malgré l'augmentation des capacités de stockage et le débit internet ; la diminution efficace des données reste d'intérêt majeur.

C'est Claude Shannon qui est à l'origine de la compression. Tout commence par un jeu de société qu'il pratiquait avec ses invités : il ouvrait un livre au hasard et commençait à lire un paragraphe et s'arrêtait. Il demandait ensuite à ses invités de deviner une à une les lettre suivantes. Ils trouvaient la lettre dans environ 75% des cas, alors Shannon déduisait que la langue anglaise possède un taux de redondance égale à 75%. [1]

Les données que nous manipulons (texte, son, image) n'ont pas la même probabilité d'apparition de symboles et leur apparition n'est pas uniforme. C'est cette caractéristique qui insiste à compresser les données et c'est elle qui permet, souvent de réussir.

Définition de la compression

La compression de données : c'est l'ensemble des méthodes que l'on utilise pour réduire la taille physique de blocs d'informations sans perte d'information importante.

Les différents algorithmes de compression sont basés sur 3 critères :

- Le taux de compression : c'est une mesure de la performance d'un algorithme de compression, il est généralement exprimé en pourcentage. Le taux de compression est le rapport entre la taille des données originales et celle de données compressées. Plus le taux de compression est élevé, plus la taille du fichier compressé est faible, plus la compression est performante.

La formule correspondante est :

$$taux\ de\ compression = 1 - \frac{taille\ du\ fichier\ compressé}{taille\ du\ fichier\ initial}$$

- La qualité de compression : sans ou avec pertes (en donnant le pourcentage de perte).
- La vitesse de compression et de décompression.

Un compresseur utilise un algorithme qui sert à optimiser les données en fonction du type de données à compresser ; un décompresseur est donc nécessaire pour reconstruire les données grâce à l'algorithme dual de celui utilisé pour la compression.

La méthode de compression dépend du type de données à compresser car une image ou un fichier audio ne représentent pas le même type de données.

Intérêts de la compression

De nos jours, la puissance des processeurs augmente plus vite que les capacités de stockage, et énormément plus vite que la bande passante des réseaux.
Il y a donc un déséquilibre entre le volume des données qu'il est possible de traiter, de stocker, et de transférer.
Et c'est pour cette raison qu'il faut penser à la réduction de la taille des données.

La compression de données est un atout qui nous permet de garantir les deux critères qu'on doit toujours prendre en compte en informatique, à savoir le temps et l'espace mémoire.

- Le temps : grâce à la compression on peut diminuer le temps de transfert des données dans les réseaux câblés ou sans fil de façon proportionnelle au taux de compression.

- L'espace mémoire : la compression est un outil très efficace qui nous offre la possibilité de gagner de l'espace dans les supports de stockage de données.

Les formats compressés sont partout présents. Un DVD contient une vidéo compressée. Un appareil photo stocke les images dans un format compressé. Un lecteur audio contient des titres dans un format compressé également. Un téléphone portable lui aussi compresse les communications audio avant de les transmettre à la tour GSM. Les pages web sont transmises dans un format compressé au navigateur. Toutes ces applications seraient infaisables sans algorithmes de compression : en effet, la taille des données brutes, c'est-à-dire non compressées, est tout simplement trop grande pour être stockée sur un DVD, dans un lecteur audio ou pour être transmises à un débit raisonnable à travers un réseau.

Classification des algorithmes de compression

Dans le domaine de la compression, il existe plusieurs façons de comparer les types de compression. Pour cette raison, on va montrer comment les algorithmes de compression sont classifiés.

Compression symétrique / asymétrique :

Compression symétrique : la même méthode est utilisée pour compresser et décompresser l'information, il faut donc la même quantité de travail pour chacune de ces opérations. C'est ce type de compression qui est généralement utilisé dans les transmissions de données.

Compression asymétrique : l'algorithme de décompression est différent de celui de compression, ce qui peut être exploité pour avoir un algorithme de compression performant et un algorithme de décompression rapide. Le temps de compression est différent du temps de compression.

Compression physique / logique :

Compression physique : elle est exécutée exclusivement sur les informations contenues dans les données. Cette méthode produit typiquement des résultats incompréhensibles qui apparemment n'ont aucun sens. Le résultat d'un bloc de données compressées est plus petit que l'original car l'algorithme de compression physique a retiré la redondance qui existait entre les données elles-mêmes.

La compression logique : elle est accomplie à travers le processus de substitution logique qui consiste à remplacer une information par une autre information équivalente en gardant la même signification
Exemple : Changer « United Nations » en UN.

Compression statistique / numérique:

Compression statistique : dans ce type la valeur des motifs ne compte pas. Ce sont les probabilités qui comptent, et le résultat est inchangé par substitution des motifs.

Compression numérique : les valeurs des motifs influent sur la compression (par exemple JPEG), et les substitutions sont interdites.

Compression sans perte/avec perte :

Compression sans perte : (non *destructive*, réversible ou conservative) la compression est dite sans perte lorsqu'il n'y a aucune perte de données sur l'information d'origine. Il y a autant d'informations après la compression qu'avant, elle est seulement réécrite d'une manière plus concise. La compression sans perte est dite aussi compactage.

Le taux de compression des algorithmes sans perte est en moyenne de l'ordre de 40% pour des données de type texte. Par contre, ce taux est insuffisant pour les données de type multimédia. Il faut donc utiliser un nouveau type de compression pour résoudre ce problème qui est la compression avec perte.

Compression avec perte : (destructive, irréversible ou non conservative), La compression avec pertes ne s'applique qu'aux données « perceptuelles », en général sonores ou visuelles, qui peuvent subir une modification, parfois importante, sans que cela ne soit perceptible par un humain. La perte d'information est irréversible, il est impossible de retrouver les données d'origine après une telle compression.

Cette classification (sans ou avec perte) est le critère de comparaison le plus important pour les algorithmes de compression puisque ce dernier influence sur la qualité de la compression, et c'est d'ailleurs pour cette raison qu'on va séparer les chapitres de ce mémoire en fonction de ce critère.

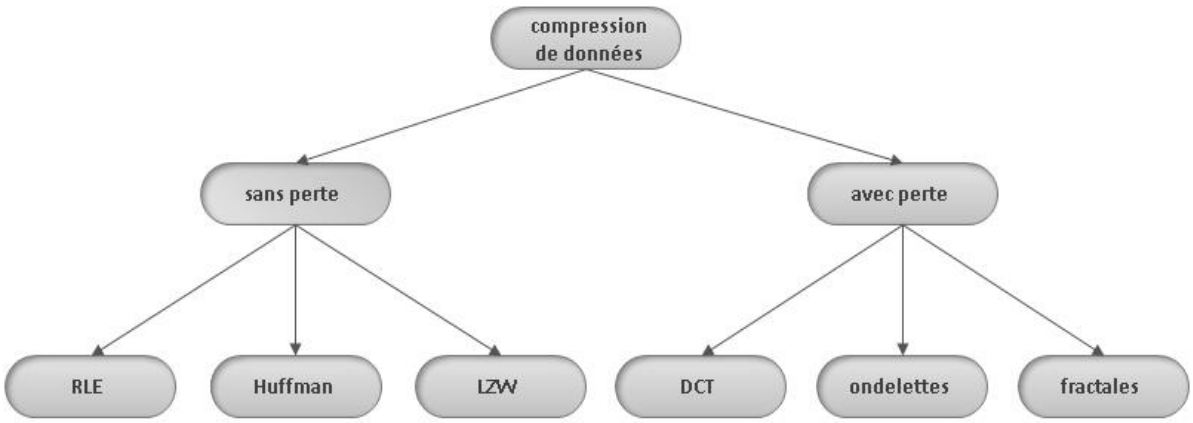


Figure 5: Schéma récapitulatif des méthodes de compression abordée dans le mémoire

Chapitre 2

Compression sans perte

Introduction

La compression sans perte, comme son nom l'indique, elle garde l'information et réduit seulement la taille, elle est aussi connue sous le nom réversible ou conservative. Les données en entrée sont récupérable entièrement. Les méthodes abordées dans ce chapitre sont :

- Run-length Encoding (RLE)
- Huffman
- Lempel Ziv Welsh (LZW)

Méthode RLE

Introduction

Le Run-Length Encoding est un algorithme de compression basé sur les répétitions. Il s'applique principalement sur les informations ayant un bon nombre de redondances. Il est relativement pratique dans les formats d'images en noir et blanc représentés sous forme bitmap.

Principe

Le RLE est basé sur le fait que dans un fichier on peut trouver une suite de bits identiques et consécutifs. Le principe consiste à donner le nombre de redondances d'un bit suivi de ce bit.

Exemple

Dans la suite de caractère suivant « AAAADDEBBBBB » nous avons 12 caractères. Après compression on obtient la chaine « 4A2D1E5B » contenant 8 caractères, donc un gain de 4 caractères (4 octets). Dans un autre exemple on considère la suite suivante « compression » ayant 11 caractères. Après compression on obtient « 1c1o1m1p1r1e2s1i1o1n » ce qui donne 20 caractères donc une perte de 9 caractères (9 octets).

Nous constatons alors que cette méthode n'est pas efficace dans certains cas où il n y a pas un nombre de redondances important.

Algorithme

Pour procéder à la compression on répète les étapes suivantes jusqu'à atteindre la fin du fichier :

1. Lire un bit
2. Si le bit est identique au précédent alors incrémenter le compteur
3. Si le bit est différent alors afficher la valeur du compteur suivi de ce bit puis le réinitialiser.

Pour décompresser il suffit de lire le compteur et afficher l'information suivante autant de fois que la valeur du compteur. Répéter la procédure jusqu'à fin du fichier.

Implémentation

Prenons l'implémentation de l'algorithme RLE réalisé par *Robert Sedgewick et Kevin Wayne* [2] qui est effectué au niveau de la représentation binaire des données, en utilisant l'API suivante (l'implémentation complète est dans l'annexe 1):

- expand : classe qui réalise la décompression en lisant du fichier compressé un octet qui représente le compteur puis écrire le bit courant autant de fois que la valeur du compteur puis passer à l'autre bit, ainsi de suite jusqu'à atteindre la fin du fichier.
- Compress : classe qui effectue la compression en parcourant le fichier original bit par bit jusqu'à sa fin, si le bit lus est différent du précédent alors écrire la valeur du compteur et le remet à 0 puis l'incrémenter, sinon incrémenter le compteur.

Expérimentation

Nous avons exécuté cet algorithme sur différents exemples :

1. L'exemple '' To be or not to be ''

Nous avons dans ce fichier 152 bits et après exécution nous avons obtenus 584 bits ce qui fait une grande perte d'espace, donc cet algorithme n'est pas du tout performant dans ce cas.

2. Essayer l'exécution sur un texte (annexe 2) ça donne :

Au départ, nous avons dans le fichier 10568 bits et après compression nous avons obtenus 44440 bits. Même dans ce cas là l'algorithme ne donne pas de bon résultat.

3. Exécution sur une l'image bitmap (noir et blanc) suivante:

Nous avons dans cette image 23040432 bits et après exécution nous avons obtenus un fichier compressé qui contient 12799400 bits. Ce qui donne le taux de compression 44%.

Nous constatons que cette méthode est plus efficace pour les images (Bitmap) que pour les textes. Donc l'efficacité de l'algorithme RLE dépend du type de données à compresser.

Analyse de l'algorithme

La complexité de l'algorithme de compression est toujours linéaire vu qu'il utilise une seule boucle, donc il est dans l'ordre O(N), avec N qui représente le nombre de données.

D'après l'exemple précédent, Nous constatons que cet algorithme est plus efficace pour les images (Bitmap) grâce aux pixels redondants dans les surfaces ayant la même couleur, par contre il est moins efficace pour les textes car les redondances sont rares. Donc l'efficacité de l'algorithme RLE dépend fortement du type de données à compresser.

Applications

L'algorithme RLE intervient principalement dans les données ayant un bon nombre de répétitions successives comme les images Bitmap (notamment celles en noir et blanc), il est utilisé aussi pour la transmission par fax et dans les formats JPEG et PCX.

Conclusion

La compression RLE est donc efficace sur les données ayant des informations redondantes, elle présente l'avantage d'être facile à manipuler. Les méthodes qui vont suivre, Huffman et LZW, s'appliqueront sur des données plus générales.

Méthode Huffman

Le codage de Huffman se fait au niveau de la représentation binaire, son principe est de réduire le nombre de bits en codant les octets les plus fréquents par des mots courts et les moins fréquents par des mots plus longs, il est sans perte. Cet algorithme a été inventé en 1952 par David Huffman et est encore utilisé de nos jours.

Le codage

Le codage se fait en deux parties : construction de l'arbre de Huffman permettant d'attribuer un code pour chaque octet, ensuite le parcours du document pour le coder selon les codes associés.

Construction de l'arbre

1. Calculer la fréquence d'apparition de chaque caractère.
2. Associer un nœud à chaque octet dont le poids est sa fréquence.
3. Classer les nœuds dans un ordre croissant de leurs poids.
4. Extraire les deux nœuds ayant le poids minimum.
5. Attacher les deux nœuds avec une racine qui prendra comme poids la somme des poids de ses deux fils.
6. Répéter à partir de l'étape 3 jusqu'à l'obtention d'un seul arbre binaire qui représente l'arbre de Huffman.

Ensuite pour attribuer le nouveau code pour chaque donnée, on démarre de la racine arrivant à cette donnée (en octet), la longueur du nouveau code est celle du chemin.

Pour atteindre le fils gauche on ajoute un 0 au code et pour le fils droit on ajoute un 1.

Ce codage est dit préfixe car chaque information est représentée par un mot différent d'un autre et chaque mot ne peut être un préfixe d'un autre mot, le codage est donc unique et il n'y a pas d'ambiguïté pour décoder l'information.

Le décodage

A partir de la liste de bits du code on reconstitue les données originales par un parcours descendant dans l'arbre de Huffman en partant de la racine et suivant les arêtes selon les bits du code jusqu'à atteindre une feuille, on remplace le code par la donnée dans la feuille.

On continue ainsi jusqu'à atteindre la fin de la liste de bits du code.

Compression

Pour la compression de données, on applique l'algorithme de Huffman afin d'avoir l'arbre binaire de Huffman. Puis on crée un nouveau fichier contenant deux parties :

- L'entête : il contient le nom original du fichier, la taille originale et l'arbre binaire de Huffman calculé précédemment.
- Les données compressées : dans cette partie chaque octet des données originales est substitué par son code binaire associé. [3]

Décompression

Pour la décompression : il faut d'abord récupérer depuis l'entête du fichier compressé l'arbre de Huffman, puis, en parcourant les données compressées, substituer chaque suite de bits correspondant à un chemin dans l'arborescence de Huffman par le symbole équivalent qu'on pourra écrire dans le fichier de restitution. [3]

Exemple

Prenant un exemple de codage de Huffman sur le texte suivant : To be or not to be.

En utilisant ASCII on obtient la suite de bits suivante :

01010100011011110010000001100010011001010010000001101111011100100010000001
10111001101111011101000010000001110100011011110010000001100010011001010010
1110

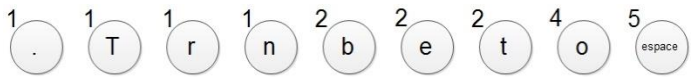
On obtient un code constitué de 152 bits. Trouvons ensuite le code Huffman associé.
Table des fréquences :
Les fréquences d'apparition de chaque caractère sont les suivantes

Tableau 3: table des fréquences

Caractère	fréquence
T	1
o	4
b	2
e	2
r	1
n	1
t	2
.	1
Espace	5

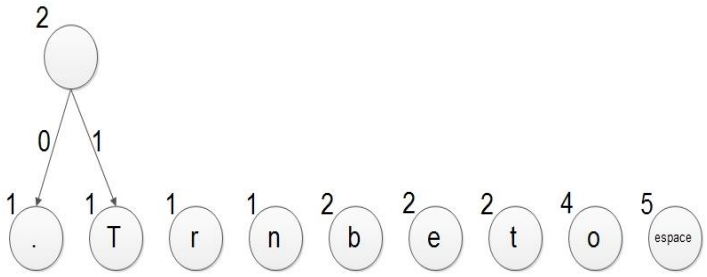
construction de l'arbre :

1. Classer les nœuds selon leurs poids.



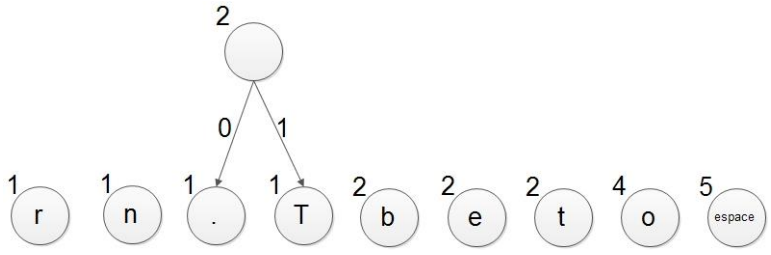
Caractère	fréquence	codage
T	1	
o	4	
b	2	
e	2	
r	1	
n	1	
t	2	
.	1	
espace	5	

2. Attacher les nœuds de poids minimums à une racine et lui associer comme poids la somme des poids de ses deux fils .



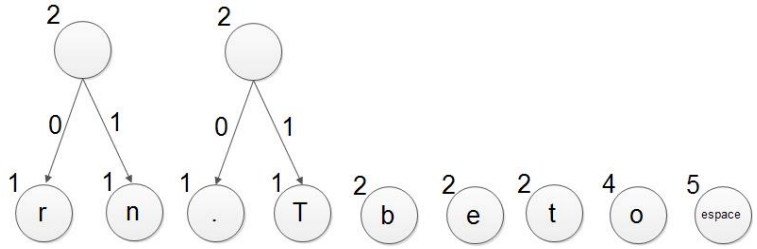
Caractère	fréquence	codage
T	1	1
o	4	
b	2	
e	2	
r	1	
n	1	
t	2	
.	1	0
espace	5	

3. Reclasser les nœuds



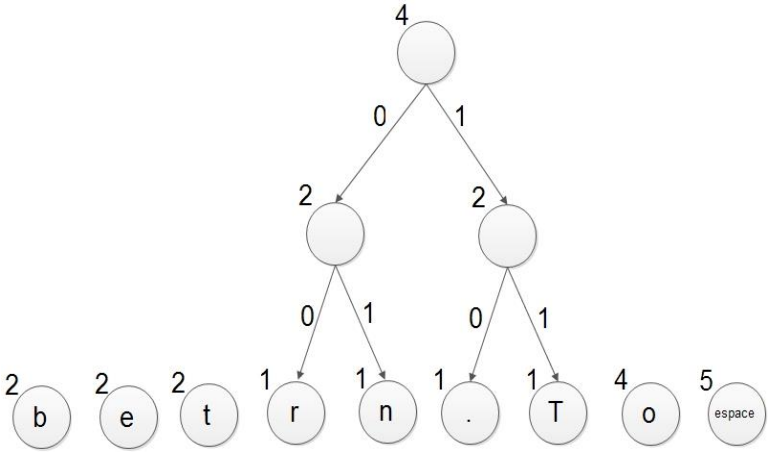
Caractère	fréquence	codage
T	1	1
o	4	
b	2	
e	2	
r	1	
n	1	
t	2	
.	1	0
espace	5	

4. Reprendre les étapes 2 et 3



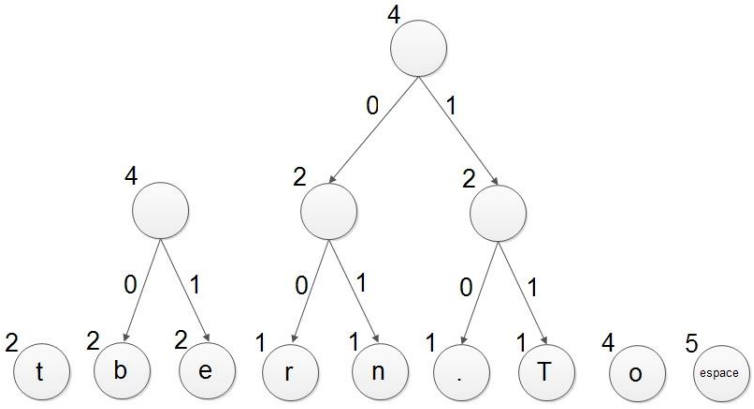
Caractère	fréquence	codage
T	1	1
o	4	
b	2	
e	2	
r	1	0
n	1	1
t	2	
.	1	0
espace	5	

Nous pouvons remarquer que les nœuds des caractères les moins fréquents s'éloignent de la racine

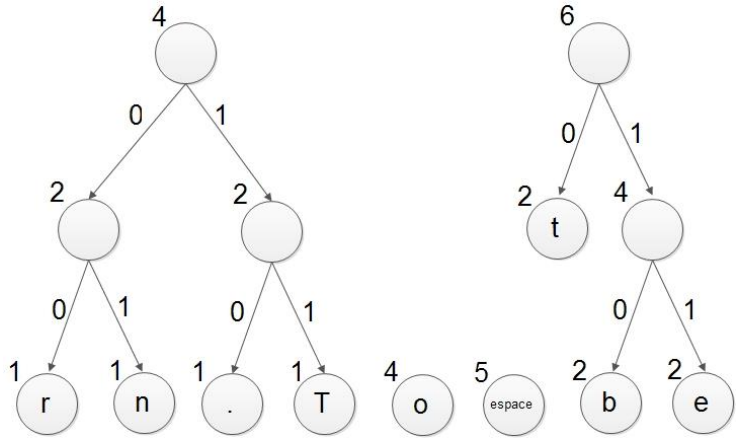


Caractère	fréquence	codage
T	1	11
o	4	
b	2	
e	2	
r	1	00
n	1	01
t	2	
.	1	10
espace	5	

Reprendre les mêmes étapes (attacher les nœuds et les reclasser)

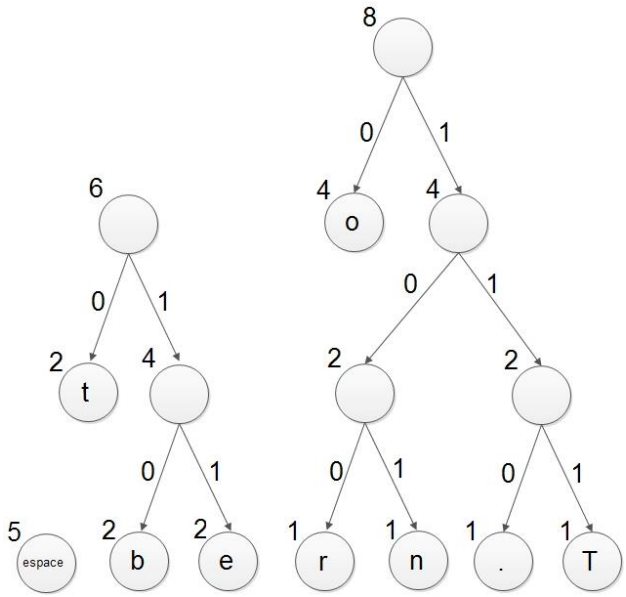


Caractère	fréquence	codage
T	1	11
o	4	
b	2	0
e	2	1
r	1	00
n	1	01
t	2	
.	1	10
espace	5	

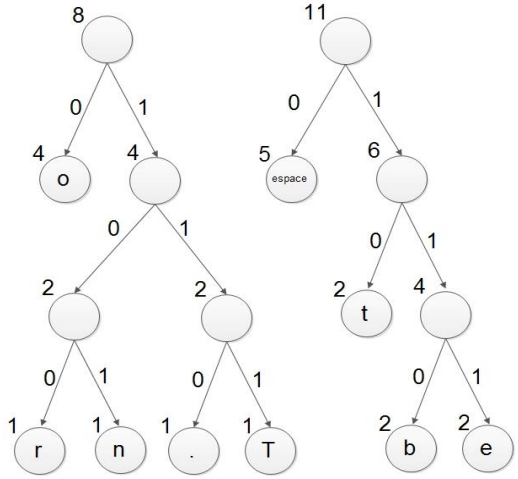


Caractère	fréquence	codage
T	1	11
o	4	
b	2	10
e	2	11
r	1	00
n	1	01
t	2	0
.	1	10
espace	5	

Continuer la construction de l'arbre jusqu'à l'obention d'un arbre complet.



Caractère	fréquence	codage
T	1	111
o	4	0
b	2	10
e	2	11
r	1	100
n	1	101
t	2	0
.	1	110
espace	5	



Caractère	fréquence	codage
T	1	111
o	4	0
b	2	110
e	2	111
r	1	100
n	1	101
t	2	10
.	1	110
espace	5	0

Quand la construction de l'arbre est terminée nous pouvons extraire le code de chaque caractère en suivant le chemin depuis la racine jusqu'à arriver à ce caractère.

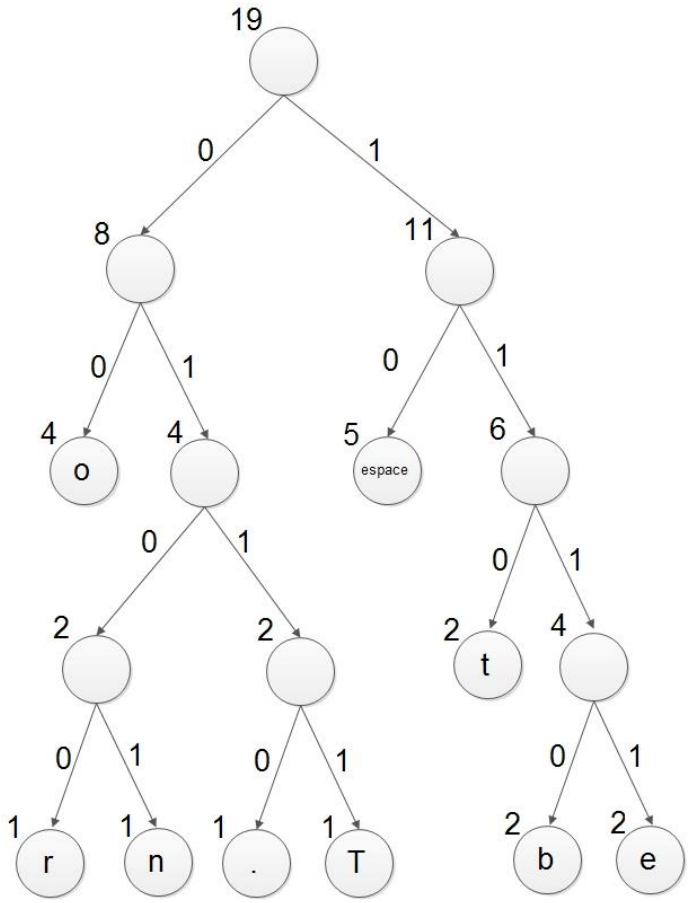


Figure 6: Arbre final de Huffman

Tableau 4:Table de codes associés à chaque caractère

Caractère	Fréquence	Code
T	1	0111
o	4	00
b	2	1110
e	2	1111
r	1	0100
n	1	0101
t	2	110
.	1	0110
Espace	5	10

Texte codé :

00110110111011111001000010000101110101100110111011111

Nombre de bits au départ été 152 bits, après compression nous avons 52 bits, ce qui fait un taux de compression de :

$$\text{Taux} = 1 - \frac{taille\ après\ compression}{taille\ originale}$$

$$= 1 - \frac{52}{152} = 0.65$$

Nous avons gagné 65 % en espace ! (sans considérer l'espace que prend l'arbre)

Implémentation

Considérons l'implémentation de l'algorithme de Huffman réalisée par *Robert Sedgewick et Kevin Wayne* [2] en utilisant l'API suivante (annexe 3) :

- BuildTrie : reçoit en entrée les fréquences de chaque caractère et construit un arbre binaire en utilisant une file de priorité.
- WriteTrie : méthode récursive qui prend en entrée un nœud et écrit 1 si c'est une feuille, suivi de la représentation binaire du caractère dans la feuille, sinon, si c'est un nœud interne, elle écrit 0 et rappelle la fonction pour ses deux fils, ceci est le parcours préfixe de l'arbre.
- ReadTrie : méthode récursive qui renvoie l'arbre de Huffman depuis sa représentation binaire.
- BuildCode : méthode récursive qui associe à chaque caractère son code en parcourant l'arbre.
- Compress : réalise la compression en calculant d'abord la fréquence de chaque caractère ensuite elle appelle la fonction BuildTrie puis la fonction BuildCode et enfin WriteTrie, et finalement coder le texte.
- Expand : réalise la décompression en appelant la fonction ReadTrie puis parcourir le code et utiliser l'arbre résultat du ReadTrie pour reconstruire le texte originale.

Expérimentation

On exécute cet algorithme sur deux exemples :

1. Notre example (To be or not to be.):

Nous avons au départ 152 bits, et après exécution on obtient 184 bits (y compris les bits du code et les bits de l'arbre).

Dans ce cas, l'algorithme ne donne pas de résultats satisfaisants.

2. Sur un texte (annexe 2) ayant 1321 caractères codé sur 10568 bits :

Après compression on obtient un fichier sur 6416 bits. Ce qui donne un taux de compression égale à 39%.

Analyse de l'algorithme

Soit R la taille du fichier original, et N le nombre de caractères distincts dans ce fichier.

La 1ère passe : parcours du fichier pour calculer les fréquences et construire l'arbre prend un temps O(R).

La 2ème passe : codage de chaque caractère selon l'arbre de Huffman prend un temps

O(N log N).

Le temps d'exécution est O (R + N log N) ≈ O (N log N). [2]

Cet algorithme est peu efficace quand il s'agit d'un grand ensemble d'informations avec peu de répétition, le N est donc grand et l'arbre occupe un grand espace mémoire.
Par contre, il est plus efficace dans le cas où les données sont répétées plusieurs fois.

Applications

L'algorithme de Huffman peut être utilisé sur le texte, l'image, le son ou la vidéo.

Il est notamment utilisé dans les formats PDF, MP3, MPEG, JPEG ainsi il utilisé par GZIP.

Conclusion

L'algorithme de Huffman est largement répondu grâce à sa performance et son large domaine d'application malgré son ancienneté, néanmoins d'autres méthodes ont été développées telle que LZW ayant plus de performances.

Méthode LZW

Introduction

La famille des algorithmes LZ* est un ensemble de techniques de compression dites de type dictionnaire. Ce sont *Abraham Lempel* et *Jakob Ziv* qui ont conçu le premier algorithme de cette famille, en 1977 sous le nom LZ77, puis vient l'algorithme LZ78, version améliorée de LZ77. Dans les années *80, Terry Welch* modifia le LZ78 pour créer *le Lempel-Ziv-Welch ou LZW.* [3]

Ce dernier est basé sur la multiplicité des occurrences des données à compresser. Il est plus simple et plus rapide. LZW est devenu très populaire et largement utilisé.

Principe de l'algorithme

Le principe est fondé sur le fait que plusieurs séquences peuvent apparaitre plusieurs fois, l'algorithme crée donc un dictionnaire afin de remplacer les séquences répétées par leurs indices dans le dictionnaire.

Le dictionnaire est un tableau dynamique contenant initialement les caractères inferieurs à 256 (tous les caractères de la table ASCII), au fur et à mesure du parcours, des séquences sont ajoutées.

Cette méthode est symétrique, la table est construite pendant la compression et la décompression, elle présente donc l'avantage du fait qu'elle n'est pas stockée dans le fichier compressé.

Algorithmes

L'algorithme de référence est publié par Therry Welch dans un article nommé ''A Technique For Hight-performence Data Compression'' [4]

Les algorithmes suivants s'inspirent de cet article.

Algorithme de compression

```
Initialiser la table avec les caractères initiaux
w ← null
Tant qu'il existe un caractère K faire
début
        lire (K)
        si wK existe dans la table alors
                w ← wK
        sinon
                écrire (code (w))
                ajouter wK dans le dictionnaire
                w ← K
        FinSi
FinTantQue
```

L'algorithme fait en sorte qu'il cherche, dans la séquence en entrée son préfixe le plus long dans la table, une fois trouvé, il est remplacé par son code dans la table, et cette suite suivie du prochain caractère est introduite dans la table.
Exemple : Nous allons appliquer l'algorithme de compression sur la chaine « abababab »

Tableau 5: Construction du dictionnaire pour compression LZW

La chaine à coder	w	K	Action	La chaine codée
abababab	null	a	Chercher un plus long préfixe	-
abababab	a	ab	Ecrire le code de « a » et ajouter « ab » au dictionnaire	97
bababab	b	ba	Ecrire le code de « b » et ajouter ba au dictionnaire	97 98
ababab	a	ab	Chercher un plus long préfixe	97 98
ababab	ab	aba	Ecrire le code de « ab » et ajouter « aba » au dictionnaire	97 98 256
abab	a	ab	Chercher un plus long préfixe	97 98 256
abab	ab	aba	Chercher un plus long préfixe	97 98 256
abab	aba	abab	Ecrire le code de « aba » et ajouter « abab » au dictionnaire	97 98 256 258
b	b	b	Ecrire le code de « b » et terminer la compression	97 98 256 258 98

Le dictionnaire :

séquence	Code
a	97
b	98
ab	256
ba	257
aba	258
abab	259

Remarque :

La taille de l'adresse n'est pas définie au début de l'algorithme, à chaque fois qu'une nouvelle puissance de 2 est ajoutée au dictionnaire, le code reçoit un bit de plus. [3]

Dans notre exemple les nouveaux codes contiennent 9 bits.
Avant, le texte été codé sur :
8 * 8 = 64 (8 caractères représentés sur 8 bits chacun)

Le nouveau code contient :
5 * 9 = 45
Ce qui signifie un gain de 29 %

Algorithme de décompression

```
Initialiser le dictionnaire pour contenir les caractères singuliers du code ASCII
K ← lire le premier code dans le fichier
w ← Nul
Ecrire (séquence(K))
Tant que le fichier n'est pas à sa fin faire
        Début
          L ← Lire le code suivant
          Si (L appartient au dictionnaire) alors
                  S ← séquence(L)
                  Sinon
                          S ← séquence(K) + w)
          Fin si
          Ecrire (S)
          w ←  S [0]
        Ajouter (séquence(K) + w) au dictionnaire
        K ← L
Fin tan que
```

Dans cet algorithme de décompression on reconstruit le dictionnaire au fur et à mesure du parcours du fichier compressé. On lit donc le premier code du fichier compressé et on affiche la séquence associé à ce code, puis on lit le code suivant, on s'assure qu'il appartient au dictionnaire et on affiche sa séquence associée. Ensuite on ajoute dans le dictionnaire la séquence de l'ancienne adresse lue concaténée à la première lettre de la séquence suivante. On continue ainsi avec les adresses suivantes jusqu'à la fin du fichier.

Exemple : Nous allons maintenant appliquer l'algorithme de décompression sur la séquence : « 97 98 256 258 98 »

Tableau 6:Construction du dictionnaire pour décompression LZW

K	L	S	W	Action	Chaine décodée
97			nul	Ecrire «a »	**a**
97	98	B	b	Ecrire « b » et ajouter « ab » au dictionnaire	a**b**
98	256	ab	a	Ecrire « ab » et ajouter « ba » au dictionnaire	ab**ab**
256	258	aba	a	Ecrire « aba » et ajouter « aba » au dictionnaire	abab**aba**
258	98	b	b	Ecrire « b » et ajouter « abab » au dictionnaire	abababa**b**

Le dictionnaire :

code	chaine
97	a
98	b
256	ab
257	ba
258	aba
259	abab

À la fin de cet algorithme nous sommes arrivés à retrouver notre chaine originale « abababab ».

Nous constatons que le dictionnaire généré par l'algorithme de compression est le même que celui généré par l'algorithme de décompression.

Analyse de l'algorithme

L'agorithme de compression LZW contient une seule boucle parcourant le fichier. Si la taille de fichier est *n*, la complexité est égale à O (*n*).

Autres algorithmes de la famille LZ*

- LZ77

C'est un algorithme de compression par dictionnaire, il utilise une fenêtre coulissante ayant une taille précise qui se déplace dans le fichier de gauche à droite, le but est de chercher la plus longue partie déjà rencontrée dans la fenêtre par un code sous forme d'un triplet : (sa position, sa longueur, le caractère qui suit).on continue de cette façon jusqu'à la fin du fichier à compresser.

- Deflate

C'est une variante de LZ77 qui couple le codage de Huffman et l'algorithme LZ77.

- LZ78

Cet algorithme utilise un dictionnaire construit au fur et à mesure du parcours du fichier, à chaque fois on cherche la plus petite séquence qui n'existe pas dans le dictionnaire pour l'ajouter puis on écrit le code de cette séquence sous forme d'un couple (position du mot

trouvé, caractère à ajouter). Le LZW est une variante de LZ78 qui utilise un dictionnaire initialisé au départ par les caractères ASCII.

Application

L'algorithme LZW est l'une des méthodes les plus efficaces pour la compression de tous type de données, il est utilisé dans les formats d'image GIF et TIFF, dans les modems V42 bis ainsi dans le logiciel de compression compress sous UNIX.

La variante Deflate a aussi un large domaine d'application, elle est utilisée pour les formats zip, gar, png et pdf, ainsi elle est aussi utilisée par les logiciels gzip et 7zip.

Conclusion

LZW est répandue grâce à sa rapidité, l'algorithme a été peu utilisé car il a été breveté par la société Unisys jusqu'à 2004 (la date d'expiration de ce brevet), actuellement l'algorithme est librement disponible pour une utilisation publique.

Des améliorations sur cet algorithme ont été développées, ce qui a fait apparaitre des variantes de LZW tel que LZEXE qui permet la compression des fichiers exécutables.

Chapitre 3

Compression avec perte

Introduction

La compression avec perte, ne s'applique qu'aux données ''perceptuelles'' (sonores et visuelles) où, pendent la compression, quelques données sont détruites sans que cela ne soit perceptible par l'humain. L'information après décompression est différente de celle en entrée. Les méthodes abordées dans ce chapitre sont :

- Transformée en cosinus discrète (DCT)
- Transformée en ondelettes
- Compression par fractales

Compression par DCT

Introduction

La transformée en cosinus discrète (DCT) est une transformation numérique sous forme de formules mathématiques. Elle est principalement utilisée dans la compression JPEG, elle représente l'image comme ensemble de fréquences et d'amplitudes plutôt que de pixels.

Nous allons aborder l'application de la DCT dans l'algorithme JPEG, vu qu'elle représente la partie la plus importante dans la compression.

Algorithme JPEG

L'algorithme JPEG, créé en 1992, est considéré comme le plus utilisé dans la compression d'image.

La compression se fait en plusieurs étapes, voici un schéma qui récapitule le séquencement des étapes :

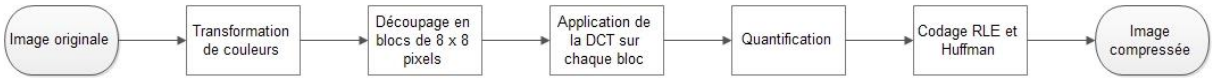


Figure 7: Schéma représentant les étapes de compression JPEG

Par la suite, on va décrire le fonctionnement de chaque étape

1. Transformation de couleurs :

   Dans cette étape, le format de couleurs RGB est converti en format YCbCr basé sur la luminance (Y) et la chrominance (différence de la couleur rouge Cr et différence de la couleur bleue Cb). Pour cause, l'œil humaine est plus sensible à la variation de luminosité qu'à la variation de couleur.

   La conversion se fait en appliquant les règles suivantes [5]:

   - Luminance = 0,29 * rouge + 0,59 * vert + 0,12 * bleu
   - Différence de rouge = rouge - luminance
   - Différence de bleu = bleu - luminance

2. Découpage de l'image

Dans cette étape, l'image est découpée en bloc 64 pixels (8 x 8 pixels) qui seront traités individuellement.

3. Application de la DCT

Avant d'appliquer la DCT il faut d'abord soustraire le nombre 128 (qui représente 2^{n-1}, n = 8 dans notre cas) de toutes les valeurs de la matrice.

Dans cette étape on applique la DCT sur chaque bloc, c'est l'étape la plus couteuse en terme de temps et de ressources utilisées, mais elle permet de séparer les basses fréquences des hautes fréquences présentes dans l'image.

La formule DCT est la suivante [6]:

$$DCT(i,j) = \frac{1}{\sqrt{2N}} C(i)C(j) \sum_{x=0}^{N-1} \sum_{y=0}^{N-1} p(x,y) \cos\left(\frac{(2x+1)i\pi}{2N}\right) \cos\left(\frac{(2y+1)j\pi}{2N}\right)$$

- N : la largeur d'un bloc, ici $N = 8$.
- i, j : les indices d'un coefficient de la DCT dans un bloc.
- x, y : les indices d'un pixel de l'image dans un bloc.
- $DCT(i,j)$: la valeur d'un coefficient dans un bloc.
- $C(x) = \frac{1}{\sqrt{2}}$ si $x = 0, C(x) = 0$ sinon.

- ***P (x, y)* : la valeur de pixel aux coordonnées (x, y)**

Dans notre cas (N = 8) ça donne :

$$G_{u,v} = \alpha(u)\alpha(v) \sum_{x=0}^{7} \sum_{y=0}^{7} g_{x,y} \cos\left[\frac{\pi}{8}\left(x + \frac{1}{2}\right)u\right] \cos\left[\frac{\pi}{8}\left(y + \frac{1}{2}\right)v\right]$$

$$\alpha\ (n) = \begin{cases} \sqrt{\frac{1}{8}}, & \text{Si } n = 0 \\ \sqrt{\frac{2}{8}}, & \text{Sinon} \end{cases}$$

- $g_{x,y}$: la valeur de pixel aux coordonnées (x, y)
- $G_{x,y}$: la valeur de la DCT aux coordonnées (x, y)

Après l'application de la DCT on arrondit les valeurs pour avoir des entiers.

Exemple :

- La matrice de pixels du bloc :

```
=

  66   84  110  167  207  209  193  178
 109  136  150  173  193  213  205  210
 137  178  194  190  185  203  223  238
 140  168  191  182  194  222  244  241
 149  153  173  186  228  245  241  226
 170  157  181  216  235  217  189  130
 191  165  205  205  180  147   82   55
 158  153  196  179  137   76   46   51
```

- La matrice après soustraction de 128 :

```
G_128 =

   -62   -44   -18    39    79    81    65    50
   -19     8    22    45    65    85    77    82
     9    50    66    62    57    75    95   110
    12    40    63    54    66    94   116   113
    21    25    45    58   100   117   113    98
    42    29    53    88   107    89    61     2
    63    37    77    77    52    19   -46   -73
    30    25    68    51     9   -52   -82   -77
```

- La matrice après application de la DCT

```
Gdct =

   358   -85  -121     6    -5     6    10     6
    66  -238    47    -1   -20   -19   -17   -13
  -188   107   -57    -8    31     1     7    -4
    19   -60   -54    75    11     7    -7     2
   -34   -47    13   -10     2    -5    -1     7
    -7    30   -12    -1    -3     7     4    -1
     3    -8    -7    -7     2     4   -13     3
     3     4     5    -4     4    -5     6    -1
```

4. Quantification

Jusqu'à cette étape, aucune information n'est perdue, on peut appliquer la DCT inversée pour récupérer toute l'information.

Le but de la quantification est d'éliminer les données à partir d'un certain seuil avec un facteur de qualité de la matrice de quantification. C'est à cette étape que la perte de données se produit.

Il existe des tables de quantification standards pour la luminance et la chrominance.

Celle de luminance par exemple est :

16	11	10	16	24	40	51	61
12	12	14	19	26	58	60	55
14	13	16	24	40	57	69	56
14	17	22	29	51	87	80	62
18	22	37	56	68	109	103	77
24	35	55	64	81	104	113	92
49	64	78	87	103	121	120	101
72	92	95	98	112	100	103	99

Le facteur de qualité de cette matrice est 50.

Voici comment obtenir une matrice de quantification avec le facteur de qualité souhaité
Si le facteur de qualité est > 50 :
Multiplier la matrice standard sur (100 – facteur de qualité)/50.
Sinon multiplier la matrice sur 50 / facteur. [7]

Remarques :

- Plus le facteur de qualité est grand plus le taux de compression est élevé, par contre, la qualité de l'image compressée va se dégrader.
- Il existe des matrices de quantification standards avec facteur de qualité égale à 50.

La matrice quantifiée est obtenue par la formule suivante :

$$Matrice\ quantifiée = DCT(i,j)/Q(i,j)$$

- DCT (i, j) : la valeur du coefficient aux coordonnées (i, j)
- Q (i, j) : la valeur de la matrice de quantification aux coordonnées (i, j)

Ensuite, les valeurs obtenues sont arrondies à l'entier le plus proche.

Exemple :

Prenons pour cet exemple une matrice de quantification avec un facteur de qualité égale à 15, cela donnera la matrice suivante :

```
table =

    53    37    33    53    80   133   170   203
    40    40    47    63    87   193   200   183
    47    43    53    80   133   190   230   187
    47    57    73    97   170   290   267   207
    60    73   123   187   227   363   343   257
    80   117   183   213   270   347   377   307
   163   213   260   290   343   403   400   337
   240   307   317   327   373   333   343   330
```

La matrice DCT quantifiée est donc la suivante :

```
     7    -2    -4     0     0     0     0     0
     2    -6     1     0     0     0     0     0
    -4     2    -1     0     0     0     0     0
     0    -1    -1     1     0     0     0     0
    -1    -1     0     0     0     0     0     0
     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0
```

5. Codage RLE et Huffman :

L'étape finale de la compression consiste à coder la matrice quantifiée par la méthode RLE puis celle de Huffman.

Applications

La compression par transformée en cosinus discrète est utilisée dans le traitement des images (comme nous l'avons vue). Une variante de DCT (DCT Modifiée) est appliquée pour la compression audio, dont les formats sont les suivants : WMA, Ogg Vorbis et MP3. La DCT est également utilisée dans le format de vidéo MPEG.

Conclusion

La DCT, comme nous l'avons abordée, est une transformation qui ne détruit pas l'information, par contre elle permet de faire un tri qui permettra ensuite la quantification et l'élimination des données les moins pertinentes à l'œil ou à l'oreille humaine.

Cette méthode est couramment utilisée, néanmoins, d'autres méthodes ont été développées dans le but de permettre un taux de compression plus élevé en gardant une bonne qualité telle que la méthode de compression par ondelettes.

Compression par ondelettes

Introduction

Le format de compression d'image le plus courant actuellement est le JPEG dont le principe est fondé sur le découpage de l'image en bloc puis l'utilisation de la DCT (Transformée en cosinus discrète), ensuite la quantification. Plus le taux de compression est élevé plus la différence entre les blocs est remarquée, ce qui rend son utilisation impossible dans l'imagerie médicale.

La FDA (Food and Drug Administration) avait interdit aux médecins de faire appel aux méthodes de compression, limitant les diagnostiques médicaux à distance jusqu'à l'apparition de la norme JPEG 2000 qui est basée sur les ondelettes.

Les ondelettes ont vu le jour vers les années 70 du dernier siècle par Jean Morlet, un ingénieur d'Elf-Aquitaine, La transformée par ondelette est aussi appelé DWT (Discrete Wavelet Transform)

Principe

Le principe des ondelettes comme la définit JM Ghidaglia dans son article ''compression odelette pour image'' est comme suit : *« La transformation en ondelettes repose non plus sur un découpage de l'image en blocs dans l'espace physique, mais sur un découpage dans l'« espace des échelles ». La notion d'échelle, essentielle en physique, fait référence au niveau auquel on perçoit un objet : par exemple, selon les problèmes étudiés, la Terre peut être considérée comme un point matériel, une sphère, un ellipsoïde de révolution, ou un solide de surface irrégulière. La transformation en ondelettes classe les échelles par ordre décroissant, et compresser par ondelettes revient à ne retenir à priori qu'un nombre déterminé d'échelles en excluant les plus fines.* » [8]

La compression par ondelette se fait en trois étapes :

- La transformation par ondelette
- La quantification
- Le codage

Compression

Nous allons nous intéresser par l'étape de la transformation par ondelette, particulièrement dans le cas d'image.

Transformée par ondelette de Haar :

1. Calcul de la moyenne entre chaque deux pixels suivant l'axe horizontale :
   $$H(x) = \frac{X_n + X_{n+1}}{2}$$
2. Calcul de l'erreur entre l'image originale et l'image sous-échantillonnée dans le sens horizontale :
   $$G(x) = \frac{X_n - X_{n+1}}{2}$$
3. Pour chacune des images obtenues, on calcule la moyenne des pixels deux à deux suivant l'axe verticale :
   $$H(y) = \frac{Y_n + Y_{n+1}}{2}$$
4. Pour chacune des images obtenues, on calcule l'erreur avec l'image originale suivant l'axe verticale :
   $$G(y) = \frac{Y_n - Y_{n+1}}{2}$$

On répète les étapes autant de fois que nécessaire pour obtenir le nombre voulu de sous-images.

Exemple :

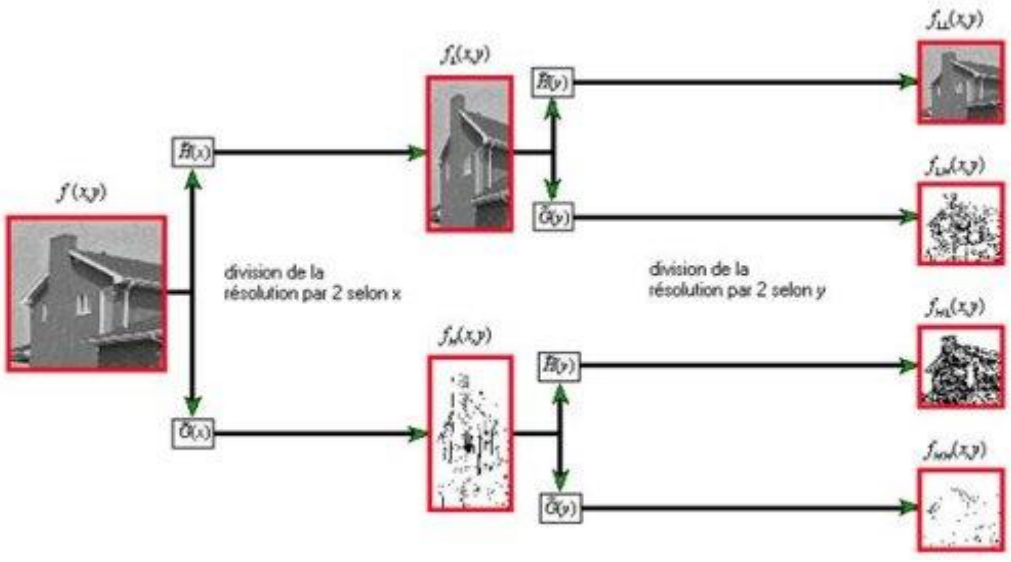


Figure 8: Exemple d'application des étapes de transformation d'une image par ondelettes

Quantification

Jusqu'à cette étape les données sont classifiées selon les fréquences. Nous savons que les hautes fréquences sont moins importantes que les basses fréquences. C'est là où intervient la quantification afin de supprimer les hautes fréquences depuis le seuil désiré.

Analyse de l'algorithme

L'algorithme est d'une complexité égale à O(N), car H et G sont à temps constant et chacune d'entre elles prend un temps O(N) lors du parcours de tous les pixels. Ce qui donne une complexité de T(N) = 2N + T (N/2), (T(N/2) car l'image sera ensuite découpée en deux à chaque itération).

Applications

L'application la plus fréquente de la compression par ondelette est dans le format d'image JPEG 2000 et le format vidéo MJPEG 2000. Elle est utilisée aussi dans l'imagerie médicale et les empreintes digitales des fichiers du FBI.

Conclusion

Les ondelettes ont un domaine d'utilisation large, et leur utilisation dans la compression offre des avantages (un taux de compression intéressant sans perdre de la qualité d'image) qui font d'elle une méthode efficace.

La méthode qui va suivre, la compression par fractale, est appliquée uniquement sur les images, elle offre le même avantage que le JPEG 2000, lors de zoom, l'image ne produit pas l'effet de pixellisation contrairement au JPEG.

Méthode fractales

Introduction

La compression fractale est un procédé qui s'applique uniquement aux images.

Le mot fractal est d'origines latines ''fractus'' qui signifie brisé, ce mot est inventé par le mathématicien français Benoit Mandelbrot à la fin des années 70 du dernier siècle qui désigne les objets irréguliers composés d'un ensemble de figures classiques telle la droite, le cercle ….etc.

En 1988, le mathématicien Michael F. Barnsley a utilisé cette propriété pour stocker les images numériques avec peu de données. Il a introduit des fonctions IFS (Iterated Function Systems) dans son livre ''Fractal everywhere'' où, selon lui, toute image peut être représentée par un ensemble d'IFS, il suffit donc de coder les IFS qui reconstituent l'image et non l'image elle-même, cela nous permet de gagner plus d'espace. [9]

Malheureusement cette méthode, même efficace, reste lente et non automatisée.

En 1992, un des élèves de Barnsley, Arnaud Jacquin a automatisé la procédure, son algorithme est encore utilisé à nos jours.

Principe

Le principe est basé sur le fait qu'une image est un ensemble de motifs identiques auxquelles on applique des transformations géométriques (rotations, translations, agrandissements, réductions).

L'image est découpée en blocs de pixels de tailles variables, ensuite il faut détecter les blocs ayant une grande similarité, on détermine les transformations nécessaires permettant de passer d'un bloc source à un autre bloc de destination telles que les rotations, réductions, réflexions, variations du contraste et de la luminosité. Il suffit par la suite de coder le bloc source et la transformation permettant d'obtenir le bloc de destination.

Compression

Voici l'algorithme qui permet la compression suivant la méthode fractale

```
DEBUT
créer un pavage de blocs source
créer un pavage de blocs destination
pour chaque bloc destination faire
        pour chaque bloc source faire
                pour toute les transformation définies faire

                        appliquer la transformation au bloc Source
                        ajuster la moyenne des couleurs des pixels
                        appliquer la réduction du bloc source vers le bloc destination
                        calculer l'erreur entre le résultat et le bloc destination

                        si l'erreur est minimale pour le bloc destination alors
                                enregistrer les modifications
                        fin si
                fin pour
        écrire dans le fichier de sortie les valeurs enregistrées
        fin pour
fin pour
FIN
```

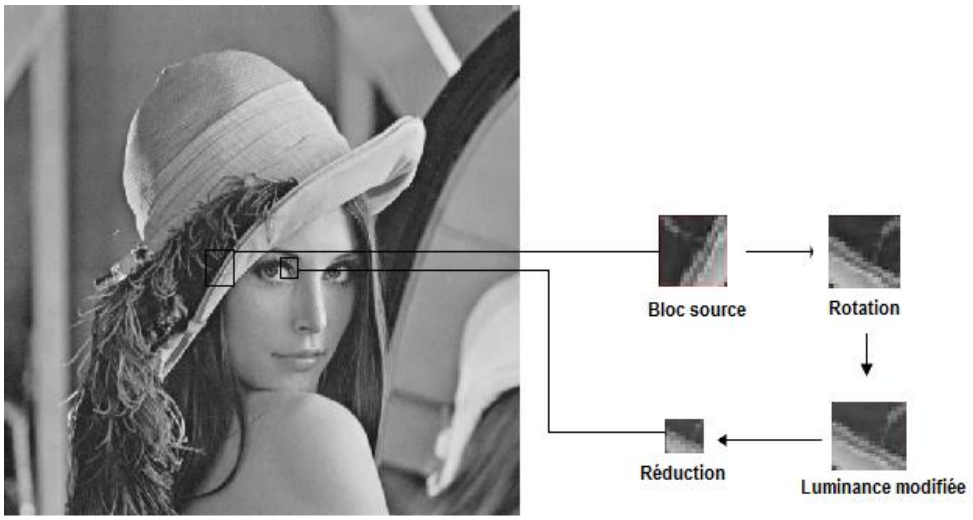


Figure 9: Schéma des étapes de compression par fractales

Décompression

La décompression consiste à lire les transformations à partir du fichier compressé et de les appliquer sur les blocs initiaux afin de retrouver l'image.

Application

Les applications des fractales se manifestent de façon importante et ce, dans de nombreux domaines : informatique, mathématique, physique, biologie…etc.

En informatique, les fractales sont utilisées dans le multimédia, elles sont exploitées dans le cinéma et les jeux vidéo pour la création de paysages naturels, arbres, montagne ainsi les objets virtuels très réalistes.

Les fractales sont utilisées dans la compression d'image, elles donnent également des résultats de haute qualité. Le format d'image FIF utilise cette méthode, il est breveté par la société ''Iterated Systems Incorporated'' fondée par Barnsley.

Avantages

Les principaux avantages de cette méthode :

- Taux de compression important
- L'image ne produit pas l'effet de pixellisation lors du zoom.

Inconvenants

- Elle est trop couteuse pendent la compression
- La méthode n'est pas normalisée, ce qui la rend inutilisable par les navigateurs web

Conclusion

La méthode fractale est très appréciée pour son taux de compression élevé, elle est néanmoins très lente en terme de temps de compression, c'est ce dont les chercheurs visent à améliorer. Ils visent aussi à introduire cette méthode dans le format vidéo et d'intégrer ce format dans les logiciels courants (navigateurs web, logiciels de retouche d'image …).

Chapitre 4

Applications

Introduction

L'application des méthodes vues dans les chapitres précédents est essentiellement liée aux types de données à compresser. Les textes et les fichiers exécutables par exemple ne tolèrent pas la perte d'informations, donc il est nécessaire d'utiliser la compression sans perte afin de garder toute l'information. Néanmoins, les données telles que l'audio, l'image ou la vidéo, peuvent être compressées avec perte en autorisant l'élimination des données imperceptibles par l'œil ou l'oreille humaine.

Domaines d'applications de la compression

Compression audio

De nombreux formats audio ont été standardisés visant à diminuer l'espace de stockage des données audio utilisant des méthodes de compression avec perte ou sans perte. Citons parmi eux, Ogg Vorbis, FLAC, WMA, AAC ou encore, le plus populaire, MP3.

La compression sans perte pour les fichiers audio est appréciée pour sa conservation de la qualité des sons, tel que le format Flac (*Free Lossless Audio Codec*) qui est libre de droit et open source et maintenu par la société Xiph.org. L'extension d'un fichier FLAC est *.flac .*

La compression destructrice, quant à elle, vise à éliminer les données sonores que l'oreille humaine ne perçoit pas comme les sons trop graves (infrasons) ou trop aigus (ultra-sons), ainsi que les sons trop faibles.

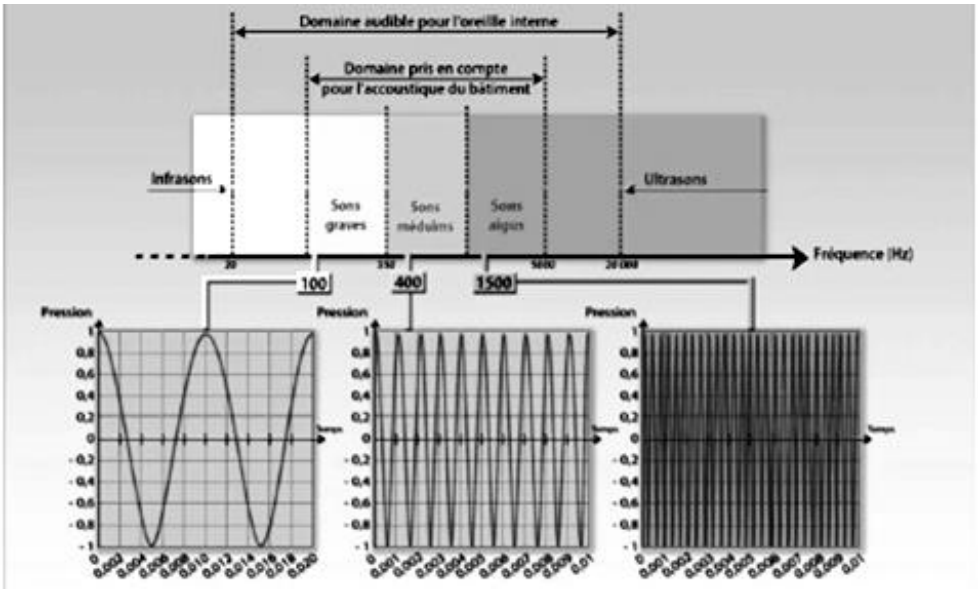


Figure 10: Schéma illustrant les domaines du son audible par l'oreille humaine

L'information originale est irrécupérable après compression.

Le format Ogg Vorbis est lui aussi développé par la société Xiph, libre de droit et open source, utilisant la transformation en cosinus discrète (DCT), ensuite, il code les données par une méthode de compression sans perte. Son extension de fichier est *.ogg* ou parfois *.oga*

Le format MP3 (*MPEG-1/2 Audio Layer 3)*, créé en 1987 par Moving Picture Experte Groupe, il utilise la transformation en cosinus discrète, puis il code les données par la méthode de Huffman. L'extension de ce format est *.mp3* .

Compression d'image

Tel que l'audio, les images existent dans différents formats et utilisant des méthodes de compression sans perte ou avec perte.

Les formats utilisant la compression sans perte permettent de garder l'information telle qu'elle est, prenant à titre d'exemple les formats PNG et GIF étant très utilisés.

Le format GIF (Graphics Interchange Format) créé en 1987 destiné à optimiser la transmission des images sur internet. Ce format utilise l'algorithme de compression LZW. Les couleurs de l'image sont indexées dans une palette (dictionnaire), chaque pixel référence l'indice de sa couleur dans la palette. L'extension des images de ce format est .gif

Le PNG (*Portable Network Graphics)* est un format d'image numérique libre de droits, créé en 1994, normalisé par l'ISO, utilisant l'algorithme de Deflate. Il a été présenté comme alternative du format GIF ayant le problème de brevet, il présente l'avantage d'en être plus performant en termes de taux de compression et la richesse de couleurs. Il s'impose comme le meilleur des formats sans perte. L'extension des images de type PNG est .png

La compression avec perte part du principe que l'œil humain n'est pas capable de voir toute les nuances, elle présente l'avantage de permettre des taux de compression élevés en éliminant les informations imperceptibles. Le JPEG est l'un des formats utilisant cette technique.

Le JPEG est un standard qui définit à la fois le format du fichier et le codec (compression/décompression), il signifie Joint Photographic Experts Group qui est le nom du groupe qui a créé cette norme au début des années 80 du dernier siècle. Il s'agit de format de compression d'image le plus utilisé dans les domaines informatiques. L'extension des images de type JPEG est .jpg

Il existe une norme JPEG-LS, c'est une méthode de compression sans perte moins connue et utilisée notamment dans l'imagerie médicale et les domaines de la recherche scientifique telles que les sciences spatiales, mais elle exige des capacités de stockage importantes.

Compression vidéo

La compression vidéo est une nécessité pour stocker ou transmettre la vidéo, vu qu'elle est gourmande en espace. Pour procéder à la compression, il convient de mettre une balance entre la qualité, la taille de fichier et le débit. Plusieurs codecs sont disponibles, aucun n'est adapté à toutes les situations, par exemple, le codec le plus performant dans les dessins animés sera peut efficace dans la compression de prises de vues réelles.

On distingue deux catégories générales dans la compression vidéo : spatiale et temporelle. La compression spatiale, appelée aussi inter-trame, consiste à appliquer la compression sur chaque trame indépendamment des autres trames. La méthode M JPEG utilise cette catégorie. [10]

La M JPEG (Motion JPEG), dont l'extension de ces fichiers est .mjpeg, utilise le principe qu'une vidéo est une succession d'images, ce codec applique l'algorithme JPEG sur chacune des images séparément. Le taux de compression obtenu n'est pas important, mais ce type offre de très bonnes conditions de travails au montage.

Figure 11: Codage Spatial

Il existe un autre format M JPEG 2000, qui part du même principe mais qui applique l'algorithme JPEG 2000 sur les images.

La compression temporelle consiste à coder une image référence. Pour les images qui se suivent, il suffit de coder les pixels dissimilaires. Le groupe MPEG a développé des standards utilisant cette technique. [10]

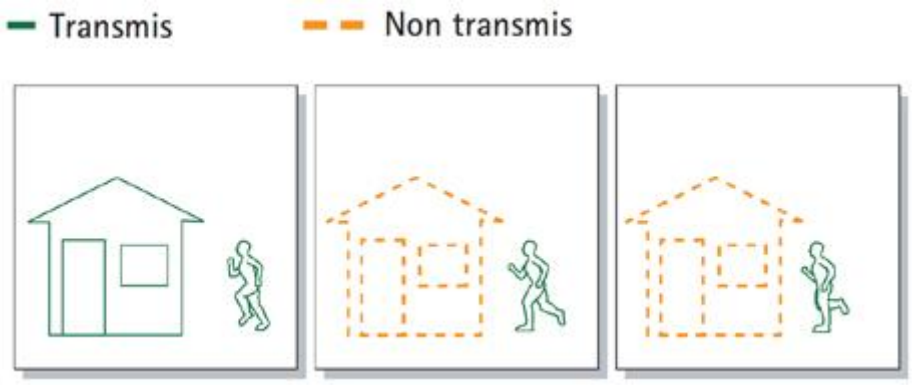


Figure 12: Codage temporel

Seule la première image est codée dans son intégralité. Les deux autres images sont des images prédites, donc on fait référence à la première image pour les éléments statiques (la maison) et on code seulement les éléments différents (l'homme qui court), ce qui permet de réduire la quantité d'information stockée et transmise.

Le groupe MPEG (Moving Picture Experts Group) a été établi en 1988 dans le but de créer des standards internationaux dans la compression, décompression, traitement et codage d'images animées et de données audio.

La norme MPEG-1 a vu le jour en 1992, elle est utilisée dans les supports CD, elle repose sur la définition de trois types d'images au sein d'une séquence vidéo :

- Les images clés (I-Frames) : sont codées séparément en utilisant JPEG
- Les images prédites (P-Frames) : sont décrites avec différences des images précédentes.
- Les images bidirectionnelles (B-frames) : sont décrites par différence avec l'image précédente et l'image suivante. [10]

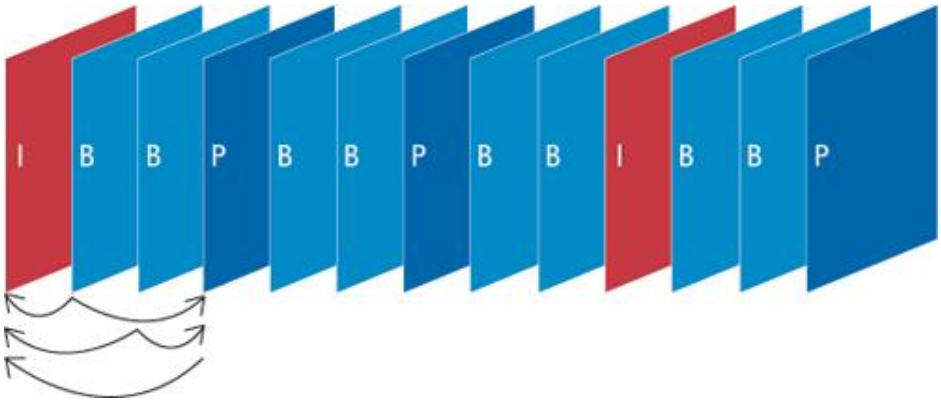


Figure 13: Norme MPEG- 1

Le MPEG-2 est principalement dédié à la télévision numérique car il a été conçu pour satisfaire des besoins en matière de taux de transfère, il est aussi utilisé dans les supports DVD.

Le format MPEG-4, quant à lui, est largement utilisé dans les nouvelles applications multimédias telle que le téléchargement, le streaming sur internet, le multimédia sur téléphone mobiles, la radio numérique, les jeux vidéos, la télévision et les supports haute définition, et pour cause, il permet de fournir des taux de compression élevés, ce qui permet le transfère d'une vidéo de bonne qualité à faible débit.

Tableau récapitulatif

Le tableau suivant regroupe quelques formats de fichiers et les méthodes de compression utilisées pour obtenir ces formats.

Tableau 7: Tableau récapitulatif

		Huffman	RLE	LZ**	DCT	ondelette	fractale
Texte et programmes		×	×	×			
Son	Aac						
	Flac						
	Mp3						
Image	Gif			×			
	Png	×		×			
	Jpeg	×	×		×		
	fif						×
	Jpeg 2000					×	
vidéo	M-JPEG	×	×		×		
	M-JPEG 2000	×	×			×	
	MPEG				×	×	

Logiciels de compression

Les logiciels disponibles en termes de compression sont nombreux, on y trouve ceux qui sont génériques (peuvent s'appliquer à plusieurs types de données), aussi, les logiciels spécifiques à certain type de données tels que les logiciels spécialisés dans la compression d'images, du son ou de la vidéo.

En matière de compression sans perte, les logiciels les plus connus sont WinZip et WinRAR, ils sont payant – mais disponibles en version de démonstration - et leurs fichiers portent l'extension .zip ou .rar, il existe plusieurs concurrents gratuits et traitent de nombreux formats, parmi ces logiciels on trouve gzip qui est largement utilisé dans les systèmes UNIX (avec l'extension .gzip pour ses fichiers) et 7-Zip (avec l'extension .7z).

Au niveau de la compression audio citons le logiciel CDex qui permet de compresser les fichiers en WAV, MP3 ou Ogg Vorbis. Il existe aussi le logiciel Ogg Drop qui permet de gérer spécifiquement le format Ogg Vorbis. Un autre logiciel très répandu, Audacity permet de mixer différents sons et aussi de les compresser dans différents formats après l'installation du plug-in associé (ces trois logiciels sont gratuits).

Pour l'image, il existe des logiciels payants telle que ''Adobe Photoshope'' et ''Paint Shope pro'' qui offre des traitements d'images ainsi que la compression dans différents formats, il existe aussi une alternative gratuite et open source, le logiciel GIMP, qui été destiné au début à Linux mais actuellement il fonctionne aussi sous Windows et Mac OS.

Pour la vidéo, on trouve le logiciel ''Adobe Premiere Pro'' qui gère de nombreux formats vidéos et qui sert aussi aux montages vidéos, le logiciel ''AVS video converter'' qui permet des conversions dans plusieurs formats, on trouve aussi le logiciel ''Rippe-it After Me'' qui, contrairement à ''Adobe Premiere Pro'' et ''AVS video converter'', est gratuit, il permet la compression/décompression des formats audio et vidéos divers (MP3, DivX et bien d'autres)

Conclusion

Finalement, nous avons vu que la compression de données ne manque pas d'applications dans notre vie quotidienne, elle est donc partout, à commencer par les domaines de communications tel que la télévision, le web ou la téléphonie, en passant par les loisirs comme les jeux vidéos, les musiques ou encore le domaine médicale et spatial. Donc on constate que sans ces applications, le multimédia (notamment les vidéos) ne serait pas accessible à tout le monde, vu son exigence d'espace important.

Conclusion

Au terme de ce travail, nous sommes arrivés à aborder quelques méthodes de compression ainsi que leurs domaines d'utilisation. Nous avons vu aussi que la compression est une nécessitée dans les réseaux et les multimédias, vu que la quantité de données est en croissance plus rapide que les supports de stockage et les bandes passantes.

Les méthodes abordées dans ce mémoire sont importantes, certaines sont très utilisées (Huffman, RLE, LZW et DCT), et d'autres moins utilisées mais considérées comme le futur de la compression d'image et de la vidéo (ondelettes et fractales).

Lors de la compression sans perte il y a des données en entrée qui restent inchangées, pour cause le comportement des algorithmes avec les données est différent, il faut donc choisir la méthode selon les propriétés des données à compresser :

- La méthode RLE s'applique sur les données ayant un grand nombre de données identiques et consécutives.
- La méthode Huffman mise sur la fréquence des données, elle permet un codage optimal mais présente l'inconvenant en terme d'espace car l'arbre doit être stocké.
- La méthode LZW exploite la répétition de mots, elle présente l'avantage de ne pas stocker le dictionnaire, donc seules les données compressées sont gardées.

Par conséquence, si on compresse les mêmes données par différentes méthodes les taux seront différents.
En terme de temps d'exécution, le RLE est le plus rapide suivit par LZW ensuite Huffman.

En revanche, le choix de la méthode de compression avec perte dépend du besoin d'utilisation notamment le besoin de qualité.

- La DCT est très utilisée car elle est la base du format JPEG qui, lui même, est très répondu grâce à son taux de compression très élevé.
- La compression par ondelettes est moins utilisée que la DCT mais elle présente un taux de compression plus élevé que la compression par DCT en gardant plus de qualité.

Autrement dit, si on compresse la même image en utilisant les deux méthodes avec un taux identique, l'image compressée par ondelettes a plus de qualité que celle compressée à l'aide de DCT.

- La compression par fractale est moins utilisée que les deux précédentes mais elle offre une bonne qualité 'image après compression.

L’apparition des nouvelles technologies offre de nouveaux domaines d’utilisation de la compression et exigent, parfois, de créer de nouvelles méthodes. A titre d’exemple, le navigateur web ‘’Chrome’’ de Google qui utilise la compression dans la livraison des pages web sur les systèmes Android et iOS.

La compression est aussi utilisée dans les GPS, les images 3D et dans, l’un des domaines majeurs d’utilisation de la compression, les bases de données.

Il existe encore d’autres méthodes telle que la compression matérielle proposée par les organismes VESA et MIPI visant à réduire la bande passante utilisée, destinée pour les ordinateurs portables, les tablettes et les Smartphones.

La compression reste encore un objet de recherche vaste visant à développer des méthodes rapides, avec des taux et des qualités élevés.

Annexes

Annexe 1 : Texte

La sécurité informatique sera prioritaire en 2014

MONTRÉAL - La sécurité informatique sera assurément un dossier prioritaire en 2014, selon la firme de solutions en sécurité Symantec.

Ainsi, les utilisateurs prendront finalement des mesures concrètes pour garantir la confidentialité de leurs données.

Symantec souligne qu'en 2013, les problèmes de confidentialité ont fait les manchettes, amenant les individus et les entreprises à prendre conscience de la quantité d'informations personnelles qui sont partagées et recueillies chaque jour par un nombre effarant de personnes.

Il est donc fort probable, selon la firme, que la protection des données confidentielles devienne une fonctionnalité à part entière des produits existants et à venir.

Parallèlement, les escrocs, les collectionneurs de données et les cybercriminels deviendront encore plus sophistiqués, et s'intéresseront au moindre réseau social, aussi obscur soit-il.

Enfin, selon Symantec, les millions d'appareils connectés à Internet, généralement avec un système d'exploitation intégré, vont attirer les pirates informatiques comme un aimant en 2014. La firme rappelle que les spécialistes de la sécurité ont déjà prouvé qu'il était possible d'attaquer les télévisions intelligentes, les équipements médicaux et les caméras de sécurité.

Annexe 2 : Implémentation de l'algorithme RLE

```
/*************************************************************************
 *  Compilation:  javac RunLength.java
 *  Execution:    java RunLength - < input.txt   (compress)
 *  Execution:    java RunLength + < input.txt   (expand)
 *  Dependencies: BinaryIn.java BinaryOut.java

*************************************************************************/

public class RunLength {
    private static final int R   = 256;
    private static final int lgR = 8;

    public static void expand() {
        boolean b = false;
        while (!BinaryStdIn.isEmpty()) {
            int run = BinaryStdIn.readInt(lgR);
            for (int i = 0; i < run; i++)
                BinaryStdOut.write(b);
            b = !b;
        }
        BinaryStdOut.close();
    }

    public static void compress() {
        char run = 0;
        boolean old = false;
        while (!BinaryStdIn.isEmpty()) {
            boolean b = BinaryStdIn.readBoolean();
            if (b != old) {
                BinaryStdOut.write(run, lgR);
                run = 1;
                old = !old;
            }
            else {
                if (run == R-1) {
                    BinaryStdOut.write(run, lgR);
                    run = 0;
                    BinaryStdOut.write(run, lgR);
                }
                run++;
            }
        }
        BinaryStdOut.write(run, lgR);
        BinaryStdOut.close();
    }


    public static void main(String[] args) {
        if      (args[0].equals("-")) compress();
        else if (args[0].equals("+")) expand();
        else throw new IllegalArgumentException("Illegal command line
argument");
    }
```

Annexe 3 : Implémentation de l'algorithme Huffman

```
/*************************************************************************
 *  Compilation:  javac Huffman.java
 *  Execution:    java Huffman - < input.txt   (compress)
 *  Execution:    java Huffman + < input.txt   (expand)
 *  Dependencies: BinaryIn.java BinaryOut.java

 *************************************************************************/

public class Huffman {

    // alphabet size of extended ASCII
    private static final int R = 256;

    // Huffman trie node
    private static class Node implements Comparable<Node> {
        private final char ch;
        private final int freq;
        private final Node left, right;

        Node(char ch, int freq, Node left, Node right) {
            this.ch    = ch;
            this.freq  = freq;
            this.left  = left;
            this.right = right;
        }

        // is the node a leaf node?
        private boolean isLeaf() {
            assert (left == null && right == null) || (left != null &&
right != null);
            return (left == null && right == null);
        }

        // compare, based on frequency
        public int compareTo(Node that) {
            return this.freq - that.freq;
        }
    }


    // compress bytes from standard input and write to standard output
    public static void compress() {
        // read the input
        String s = BinaryStdIn.readString();
        char[] input = s.toCharArray();

        // tabulate frequency counts
        int[] freq = new int[R];
        for (int i = 0; i < input.length; i++)
            freq[input[i]]++;

        // build Huffman trie
```

```
Node root = buildTrie(freq);

        // build code table
        String[] st = new String[R];
        buildCode(st, root, "");

        // print trie for decoder
        writeTrie(root);

        // print number of bytes in original uncompressed message
        BinaryStdOut.write(input.length);

        // use Huffman code to encode input
        for (int i = 0; i < input.length; i++) {
            String code = st[input[i]];
            for (int j = 0; j < code.length(); j++) {
                if (code.charAt(j) == '0') {
                    BinaryStdOut.write(false);
                }
                else if (code.charAt(j) == '1') {
                    BinaryStdOut.write(true);
                }
                else throw new IllegalStateException("Illegal state");
            }
        }

        // close output stream
        BinaryStdOut.close();
    }

    // build the Huffman trie given frequencies
    private static Node buildTrie(int[] freq) {

        // initialze priority queue with singleton trees
        MinPQ<Node> pq = new MinPQ<Node>();
        for (char i = 0; i < R; i++)
            if (freq[i] > 0)
                pq.insert(new Node(i, freq[i], null, null));

        // merge two smallest trees
        while (pq.size() > 1) {
            Node left  = pq.delMin();
            Node right = pq.delMin();
            Node parent = new Node('\0', left.freq + right.freq, left,
right);
            pq.insert(parent);
        }
        return pq.delMin();
    }


    // write bitstring-encoded trie to standard output
    private static void writeTrie(Node x) {
        if (x.isLeaf()) {
            BinaryStdOut.write(true);
            BinaryStdOut.write(x.ch, 8);
            return;
        }
```

```
        BinaryStdOut.write(false);
        writeTrie(x.left);
        writeTrie(x.right);

    }

    // make a lookup table from symbols and their encodings
    private static void buildCode(String[] st, Node x, String s) {
        if (!x.isLeaf()) {
            buildCode(st, x.left,  s + '0');
            buildCode(st, x.right, s + '1');
        }
        else {
            st[x.ch] = s;
        }
    }


    // expand Huffman-encoded input from standard input and write to
standard output
    public static void expand() {

        // read in Huffman trie from input stream
        Node root = readTrie();

        // number of bytes to write
        int length = BinaryStdIn.readInt();

        // decode using the Huffman trie
        for (int i = 0; i < length; i++) {
            Node x = root;
            while (!x.isLeaf()) {
                boolean bit = BinaryStdIn.readBoolean();
                if (bit) x = x.right;
                else     x = x.left;
            }
            BinaryStdOut.write(x.ch, 8);
        }
        BinaryStdOut.close();
    }


    private static Node readTrie() {
        boolean isLeaf = BinaryStdIn.readBoolean();
        if (isLeaf) {
            return new Node(BinaryStdIn.readChar(), -1, null, null);
        }
        else {
            return new Node('\0', -1, readTrie(), readTrie());
        }
    }


    public static void main(String[] args) {
        if      (args[0].equals("-")) compress();
        else if (args[0].equals("+")) expand();
        else throw new IllegalArgumentException("Illegal command line
argument");
    }

}
```

Bibliographie

[1] F. Garagne, C. Knoff et G. Lecourtois, Codage, Compression et Cryptologie, 2005.

[2] R. Sedgewick et K. Wayne, Algorithms Fourth edition, Addison-Wesley.

[3] V. PEREIRA, F. LEPRETTE et V. HACAULT, Compression de données, 2004.

[4] T. A. Welsh, «A Technique For Hight-performence Data Compression,» juin 1984.

[5] C. Flament et L. Magain, «Algorithmes de compression: compression d'image: compression JPEG,» juin 2006. [En ligne]. Available: http://www.ulb.ac.be/cours/acohen/travaux_2006_infodoc/CompressionNumerique/TypeDonneesImageJPEG.htm. [Accès le 2013].

[6] S. Ouchraa, H. Badri et Z. Drissi El maliani, «Chapitre3. principe,» 2010. [En ligne]. Available: http://www.imagenumerique.50webs.com/transformee%20DCT.html. [Accès le 2013].

[7] K. Cabeen et P. Gent, «Image Compression and The Descrete Cosine Transform».

[8] J.-M. Ghidaglia, «Des ondelettes pour compresser les images,» *La recherche,* 1 janvier 2002.

[9] A. Gadel, «DJNet Developpers - Les techniques de compression: les fractales,» 26 octobre 2001. [En ligne]. Available: http://www.journaldunet.com/developpeur/tutoriel/gra/011026gra_fractales.shtml. [Accès le 2014].

[10] «Compression vidéo - Guide de technique,» [En ligne]. Available: http://www.axis.com/fr/products/video/about_networkvideo/compression.htm. [Accès le 2014].