

REPUBLIQUE ALGERIENNE DEMOCRATIQUE ET POPULAIRE

**MINISTERE DE L'ENSEIGNEMENT SUPERIEUR ET DE LA RECHERCHE
SCIENTIFIQUE**



UNIVERSITE MOULOU D MAMMERI TIZI-OUZOU

FACULTE DE GENIE ELECTRIQUE ET INFORMATIQUE

DEPARTEMENT D'INFORMATIQUE



Mémoire

De fin d'étude

En vue d'obtention de master en informatique

Option : réseau, mobilité et système embarqué

Thème

Implémentation et évaluation d'une
nouvelle méthode de compression temps
réel basée sur les approche LZW et VLC

Proposé par :

M^r SADOU SAMIR

Réalisé par :

Mr. BELARIF Merzak

Promotion :

2012 / 2013

Remerciements

*Je remercie le Tout Puissant, qui m'a donné la force et la
patience pour*

L'accomplissement de ce modeste travail.

*Je remercie chaleureusement mon promoteur, Mr SADOU
Samir, pour m'avoir proposé ce sujet, en me faisant confiance,
ainsi que pour avoir dirigé mon travail avec ses orientations, ses
précieux conseils, ses remarques constructives.*

*Un grand Merci aux membres du jury pour l'honneur qu'ils
m'ont attribué pour évaluer et juger mon travail.*

*Merci à ma famille et mes amis pour leurs soutiens et leurs
encouragements.*

*Et à tous ceux qui de près ou de loin m'ont aidés à
l'aboutissement de ce travail.*

Merci infiniment à tous

Merzak.B

Dédicaces

Je dédie ce modeste travail à :

*Ma grand-mère Aïcha, mon grand-père mohamed pour lequel je
prie pour que dieu l'accueille dans son vaste paradis*

*Mon grand-père Slimane, ma grand-mère Djouhar pour laquelle
je prie pour que dieu l'accueille dans son vaste paradis*

Mes très chers parents

Mon petit frère Rafik

*Mes sœurs : Karima, Kahina, Faroudja, Fazia, Aldjia et
Nassima*

Mes beaux-frères : Rachid, Mouloud et Rabah

A toute ma famille

*A tous ceux qui m'ont aidé de près ou de loin dans mes
études*

A ma très chère amie Hakima

A mes amis : Aïssa, Yacine, Samir, Yazid, et Karim

A tous mes proches grands et petits

Merzak, B

Table des matières

Introduction Générale	1
-----------------------------	---

Chapitre I Généralités

1. Introduction	3
2. Introduction à la théorie de l'information	3
3. Définition et intérêt de la compression	4
3.1. Définition	4
3.2. Intérêt	4
3.3. Quelques exemples d'utilisation	4
3.4. Comment choisir un algorithme de compression	4
4. Compression sans perte (Lossless compression)	5
4.1. Quelques domaines d'application	5
4.2. Types d'algorithmes de compression sans perte	6
4.2.1. Méthode statistique	6
4.2.1.1. Codage de Huffman	6
4.2.1.2. Codage de Shannon-Fano	8
4.2.2. Méthodes de compression avec mémoire	9
4.2.2.1. Codage arithmétique	9
4.2.2.2. Codage à base d'un dictionnaire (compression substitutionnelle)	10
1) LZ77 (Lempel-Ziv 1977)	10
2) LZ78 (Lempel-Ziv 1978)	12
3) LZW (Lempel-Ziv-Welch)	12

5. Compression avec perte (Lossy compression)	14
5.1. Compression d'image fixe	15
5.1.1. JPEG	15
5.1.1.1. Séparation de l'image en macro blocs	15
5.1.1.2. Transformation des couleurs	15
5.1.1.3. sous-échantillonnage de la chrominance	16
5.1.1.4. DCT	17
5.1.1.5. Quantification	17
5.1.1.6. Codage entropique	18
5.2. compression des signaux audio	20
5.3. compression de vidéo	20
6. conclusion	21

Chapitre II La méthode LZW

1. Introduction	22
2. Définition de LZW	22
3. Historique de LZW	22
4. Caractéristiques de LZW	22
5. Principe général	23
5.1. Compression	23
5.1.1. Algorithme de compression	23
5.1.2. Exemple	24
5.2. Décompression	25
5.2.1. Algorithme de décompression	26
5.2.2. Exemple	26

6. Les variantes de LZW	28
6.1. LZW	28
6.2. LZT	28
7. Performance de LZW	28
8. Conclusion	29

Chapitre III La méthode VLC

1. Introduction	30
2. Définition	30
2.1. Codage à longueur variable (VLC)	30
2.2. Code uniquement décodable	30
2.3. Code préfixe et arbre binaire	31
2.3.1. Code préfixe	31
2.3.2. Construction du l'arbre binaire	31
3. Inégalité de Kraft-McMillan	32
4. Types de codage entropique	33
4.1. Codage statistique	33
4.1.1. Codage de Shannon-Fano	33
4.1.2. Codage de Huffman	36
4.2. Le code unaire (unary code)	39
4.3. Codage arithmétique	40
4.3.1. Codes d'Elias	40
4.3.1.1. Code Gamma γ (N)	41
4.3.1.2. Code Delta δ (N)	42

4.3.1.3. Code Omega ω (N)	44
4.3.2. Codage de Levenstein	46
5. Conclusion	47

Chapitre IV Analyse & conception

1. Introduction.....	48
2. Le code VLC proposé.....	48
2.1.Codage.....	48
2.1.1. Prefix.....	48
2.1.2. Suffix.....	49
2.2.Décodage.....	51
2.3.Longueur du code.....	51
2.4.Liste des codes VLC.....	51
2.5.Liste des codes VLC.....	53
2.6.Vérification du théorème de Kraft-McMillan.....	54
2.7.Avantage du code VLC proposé.....	55
3. Cadre de travail.....	55
4. Principe générale de la méthode LZW_VLC proposé.....	55
5. Modules et Algorithmes principaux de l'application.....	56
5.1.Algorithmes du module de compression.....	57
5.1.1. Algorithmes générale de la compression.....	58
5.1.2. Algorithme de la fonction compresser ().....	59
5.1.3. Algorithme de la fonction CoderEnVLC (Code_LZW).....	59
5.2.Algorithmes du module de décompression	63

5.2.1. Algorithme générale de la décompression.....	64
5.2.2. Algorithme de la fonction Décompresser ().....	65
5.2.3. Algorithme de la sous fonction DécodeNewCode().....	66
5.2.4. Algorithme de la sous fonction DécodeCode().....	67
5.2.5. Algorithme de la fonction DécodeEnLZW (adr_préc , adr_suiv).....	69
6. Conclusion.....	71

Chapitre V Implémentation & Evaluation

1. Introduction.....	72
2. Environnement de développement.....	72
3. Implémentation.....	72
3.1. Interface de commande.....	72
3.2. Le noyau de l'application.....	73
3.2.1. Les fonctions principales du noyau.....	73
4. Evaluation	74
4.1. Présentation de taux et temps de compression.....	74
4.1.1. Taux de compression	74
4.1.2. Temps de compression.....	78
4.2. Evaluation des résultats.....	82
4.2.1. Taux de compression.....	82
4.2.2. Temps de compression.....	82
5. Conclusion.....	82

Conclusion Générale	84
---------------------------	----

Liste des figures

Chapitre I Généralités

Figure I.1 : Schémas de compression sans perte	5
Figure I.2 : L'arbre construit selon la méthode de Huffman	7
Figure I.3 : L'arbre construit selon la méthode de Shannon-Fano	8
Figure I.4: Exemple de compression LZ77	11
Figure I.5: Exemple de décompression LZ77	11
Figure I.6 : Schémas de compression sans perte	15
figure I.7 : Conversion d'une matrice de pixel en une matrice de réel	16

Chapitre III La méthode VLC

Figure III.1: Exemple d'arbre binaire	32
Figure III.2.a: Première subdivision pour la construction du l'arbre	34
Figure III.2.b: Deuxième subdivision pour la construction du l'arbre	34
Figure III.2.c: Troisième subdivision pour la construction du l'arbre	35
Figure III.2.d: Quatrième subdivision pour la construction du l'arbre	35
Figure III.3.a: Première répartition pour la construction du l'arbre	37
Figure III.3.b: Deuxième répartition pour la construction du l'arbre	37
Figure III.3.c: Troisième répartition pour la construction du l'arbre	38
Figure III.3.d: Quatrième répartition pour la construction du l'arbre (l'arbre final)	38

Chapitre IV Analyse & conception

Figure IV.1 : Schéma générale de compression/décompression LZW_VLC.....	56
Figure IV.2 : Décomposition des modules principaux de l'application.....	57
Figure IV.3 : Organigramme illustrant les étapes de compression.....	58
Figure IV.4 : Organigramme illustrant les étapes de la fonction compresser ().....	60
Figure IV.5 : Organigramme illustrant les étapes de la fonction CoderEnVLC (Code_LZW).....	61
Figure IV.6 : Organigramme illustrant les étapes de décompression	64
Figure IV.7 : Organigramme illustrant les étapes de la fonction Décompresser ().....	66
Figure IV.8 : Organigramme illustrant les étapes de la sous fonction DécodeurNewCode ()...67	
Figure IV.9 : Organigramme illustrant les étapes de la sous fonction DécodeurCode ().....	68
Figure IV.10 : Organigramme illustrant les étapes de la fonction DécodeurEnLZW (adr_préc,adr_suiv).....	69

Chapitre V Implémentation & Evaluation

Figure V.1 : Exemple de compression avec LZW_VLC.....	73
Figure V.2 : Graphe illustrant la différence de taux de compression entre VLC, LZW et LZW_VLC des tests réalisés sur « lukas_2d_8_tif ».....	74
Figure V.3 : Graphe illustrant la différence de moyenne de taux de compression entre VLC, LZW et LZW_VLC des tests réalisés sur « lukas_2d_8_tif ».....	75
Figure V.4 : Graphe illustrant la différence de taux de compression entre VLC, LZW et LZW_VLC des tests réalisés sur « CalgaryCorpus ».....	75

Figure V.5 : Graphe illustrant la différence de moyenne de taux de compression entre VLC, LZW et LZW_VLC des tests réalisés sur « CalgaryCorpus ».....	76
Figure V.6 : Graphe illustrant la différence de taux de compression entre VLC, LZW et LZW_VLC des tests réalisés sur « LargeCanterburyCorpus ».....	76
Figure V.7 : Graphe illustrant la différence de moyenne de taux de compression entre VLC, LZW et LZW_VLC des tests réalisés sur « LargeCanterburyCorpus »...	77
Figure V.8 : Graphe illustrant la différence de moyenne de taux de compression entre VLC, LZW et LZW_VLC des tests réalisés sur les trois corpus.....	77
Figure V.9 : Graphe illustrant la différence de temps de compression entre VLC, LZW et LZW_VLC des tests réalisés sur « lukas_2d_8_tif ».....	78
Figure V.10 : Graphe illustrant la différence de moyenne de temps de compression entre VLC, LZW et LZW_VLC des tests réalisés sur « lukas_2d_8_tif ».....	78
Figure V.11 : Graphe illustrant la différence de temps de compression entre VLC, LZW et LZW_VLC des tests réalisés sur « CalgaryCorpus ».....	79
Figure V.12 : Graphe illustrant la différence de moyenne de temps de compression entre VLC, LZW et LZW_VLC des tests réalisés sur « CalgaryCorpus ».....	80
Figure V.13 : Graphe illustrant la différence de moyenne de temps de compression entre VLC, LZW et LZW_VLC des tests réalisés sur « LargeCanterburyCorpus »..	80
Figure V.14 : Graphe illustrant la différence de moyenne de temps de compression entre VLC, LZW et LZW_VLC des tests réalisés sur « LargeCanterburyCorpus »...	81
Figure V.15 : Graphe illustrant la différence de moyenne de taux de compression entre VLC,LZW et LZW_VLC des tests réalisés sur les trois corpus.....	81

Liste des tableaux

Chapitre I Généralités

Tableau I.1 : Tableau des fréquences d'apparition des symboles	7
Tableau I.2 : Tableau des fréquences d'apparition des symboles	8
Tableau I.3 : Codage de la chaîne selon la méthode LZ78	12
Tableau I.4 : Tableau représentant la compression de la chaîne donnée selon la méthode LZW	13
Tableau I.5 : Dictionnaire obtenu après compression LZW	14

Chapitre II La méthode LZW

Tableau II.1 : Tableau représentant le déroulement de la compression de la chaîne donnée selon la méthode LZW	24
Tableau II.2 : Dictionnaire obtenu après compression LZW	25
Tableau II.3 : Tableau représentant le déroulement de la décompression de la chaîne donnée selon la méthode LZW	27
Tableau II.4 : Dictionnaire obtenu après décompression LZW	27

Chapitre III La méthode VLC

Tableau III.1 : Table des fréquences des symboles	34
Tableau III.2 : Table des codes des symboles	36

Tableau III.3 : Table des fréquences des symboles	37
Tableau III.4: Code correspondant aux caractères codés	39
Tableau III.5: Exemple de code unaire	39
Tableau III.6: Codage Gamma des 20 premiers entiers positifs	41
Tableau III.7: Codage Delta des 20 premiers entiers positifs	43
Tableau III.8: Codage Omega des 20 premiers entiers positifs	45
Tableau III.9: Codage Levenstein des 10 premiers entiers positifs	46

Chapitre IV Analyse & conception

Tableau IV.1 : Tableau des codes VLC correspondants aux codes fixes sur N bits.....	52
Tableau IV.2 : Tableau des codes VLC correspondants aux codes fixes sur 4 bits.....	53

Introduction

Générale

Depuis ses débuts, l'informatique ne se contente pas seulement de résoudre des problèmes mais plutôt de les résoudre avec un maximum d'efficacité. Cette discipline connaît une véritable révolution malgré les contraintes technologiques non négligeables rencontrées tout au long de son évolution.

Vu la limite des capacités des supports de stockage et du débit des lignes de transmission, le stockage et la transmission des données ont toujours causés des problèmes à l'homme. Pour s'y confronter à ce problème, les premières solutions apportées étaient basées sur l'extension des unités de stockage et l'augmentation des vitesses de transmissions des lignes. Mais ces solutions étaient des problèmes elles-mêmes. Car, elles étaient non seulement des solutions à court terme, mais aussi très coûteuses à mettre en œuvre et peu efficaces. Mais une autre solution était proposée pour affronter ce genre de problème, cette solution est la compression.

La compression de données est un vaste sujet qui a fait l'objet de nombreux ouvrages et articles ; elle donne lieu aujourd'hui à de nombreuses recherches en raison des enjeux économiques.

C'est une technologie qui repose sur des algorithmes de codage pour faire en sorte que les données occupent le minimum d'espace pour une vitesse de restitution acceptable. Par ailleurs, il existe plusieurs critères de répartition d'algorithme. Deux grands types d'algorithmes sont à distinguer selon la perte ou non des données:

- les algorithmes où l'utilisateur cible se satisfait de recouvrir une donnée « pas très différente » de la donnée initiale c.à.d. on tolère une certaine perte de précision en échange d'un taux de compression plus élevé. Ces algorithmes sont appelés avec perte ou lossy en anglais.
- les algorithmes qui permettent à l'utilisateur-cible de recouvrer intégralement les données sans aucune perte d'information. Ces algorithmes sont appelés sans perte, conservatifs ou lossless en anglais.

La nouvelle méthode étudiée ici est basé sur deux approches qui sont LZW et VLC. En utilisant le code LZW déjà connu, et en proposant un nouveau code VLC, on essaiera de faire une sorte de double compression en temps réel. Cette nouvelle méthode est sans perte parce qu'elle assure la restitution exacte de fichier original lors de la décompression.

Et pour bien mener notre travail, nous le répartissons sur cinq chapitres :

- Chapitre I : Le premier chapitre est une introduction à la compression des données, dans laquelle nous présenterons les définitions essentielles, les différents concepts fondamentaux de la compression des données, etc...
- Chapitre II : Le second chapitre a pour objectif de présenter la méthode de compression des données LZW, ses algorithmes de compression et de décompression, quelques variantes, etc...
- Chapitre III : Le troisième chapitre a pour objectif de présenter les principales méthodes de compression des données basées sur le code VLC, ainsi que quelques algorithmes et techniques de construction de ces codes.
- Chapitre IV : Dans le quatrième chapitre, on présente d'une part, la technique de compression utilisée, les fondements et les concepts de nouveau code VLC ; et d'autre part, nous décrivons d'une façon détaillée l'analyse et la conception de la démarche adoptée de la nouvelle méthode où nous présenterons les différents modules et algorithmes nécessaires pour la mise en œuvre de l'application.
- Chapitre V : Les résultats d'implémentation et des différents tests d'évaluation de cette méthode, précédée par une présentation des différentes fonctions principales et outils nécessaires pour la réalisation de notre application, feront l'objet de ce chapitre.

Enfin, nous clôturons par une conclusion générale où nous montrerons les apports de ce travail.

Chapitre I

Généralités

1. Introduction

Vu que l'espace sur disque est précieux, la compression semble évidente pour minimiser l'espace occupé par les données sauvegardées.

Aussi le temps de transfert de l'information sur un réseau téléinformatique est tellement significatif qu'on a opté pour la compression des données qu'on envoie.

2. Introduction à la théorie de l'information

La théorie de l'information est une science débutée en 1924 sous l'égide de mathématiciens et physiciens (Gabor, Hartley, Nyquist, Wiener), elle a pris toute sa dimension avec l'élaboration de la théorie mathématique de l'information par Shannon et Weaver.

Ainsi, la théorie de l'information attribue à chaque signifiant élémentaire d'un message une mesure de sa *quantité d'information* théorique, d'autant plus élevée que ce signifiant est "nouveau", c'est-à-dire non déductible des éléments précédents. Inversement, on attribue également une *mesure de redondance* à chaque élément de message, d'autant plus faible que son poids d'information est fort.

➤ L'entropie :

soit T : un texte

$A = \{a_1, \dots, a_k\}$: l'alphabet

f_i : la fréquence d'apparition de a_i dans le texte

$|T| = \sum f_i$

$p_i = f_i / |T|$: la probabilité d'apparition de a_i

L'entropie est définie par la formule suivante :

$$H(T) = \sum_{i=1}^k p_i \left(\log_2 \frac{1}{p_i} \right)$$

- La quantité de décision D (*decision content*) est liée au nombre de symboles n parmi lesquels la source choisit ceux qu'elle va émettre : $D = \log_2(n)$
- La redondance moyenne mesure la quantité d'information non significative dans le débit de décision généré par une source, et est donc logiquement défini par $R = D - H$

3. Définition et intérêt de la compression

3.1. Définition

La compression de données est le processus qui consiste à convertir des flux de données d'entrée (données originales) dans un autre flux de données (qui a une plus petite taille). En d'autre terme, c'est l'ensemble des méthodes utilisées pour la réduction de la taille physique des données.

Ses méthodes de compression dépendent de type de données à compresser (image, audio, texte...).

Un compresseur utilise un algorithme qui sert à réduire la taille des données. Par conséquent, un décompresseur utilise un algorithme dual de celui utilisé pour la compression est indispensable pour la reconstitution des données.

3.2. Intérêt

On assiste à une croissance énorme de la puissance des processeurs par rapport à la capacité de stockage et à la bande passante des réseaux, ce qui crée un déséquilibre entre le volume de donnée qu'il est possible de traiter, de stocker et de transférer. Cela nous mène à penser que la réduction de la taille des données est indispensable en exploitant la puissance de processeurs pour pallier aux insuffisances de capacité de stockage en mémoire et des vitesses de transmission sur les réseaux.

3.3. Quelques exemples d'utilisation

- la base de données de FBI qui contient plus de 200 millions d'empreintes digitales nécessite à être compressé.
- Les Mini-Disc
- SQL Server 2012 prend en charge la compression de page et de ligne pour les tables et les index.

3.4. Comment choisir un algorithme de compression

Les critères du choix d'un algorithme de compression sont :

- Taux de compression $T = \frac{\text{taille de fichier compressé}}{\text{taille de fichier original}}$

Ou le gain de volume : $G = 1 - T$.

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

4. Compression sans perte (Lossless compression)

Les algorithmes de compression sans perte sont basés sur des techniques qui permettent la compression et la restitution de toutes les données originales lors de la décompression.

En d'autre terme c'est une technique qui permet de réduire la taille de l'information sans altération de contenu.

Ces algorithmes prennent en entrée une suite de bits X et la transformer en une suite de bits Y plus courte que X . Y sera stockée ou transmise.

En appliquant un algorithme de décompression, qui prend en entrée la suite de bits Y , on aura une suite de bits X exactement la même avec celle qu'on a compressé.

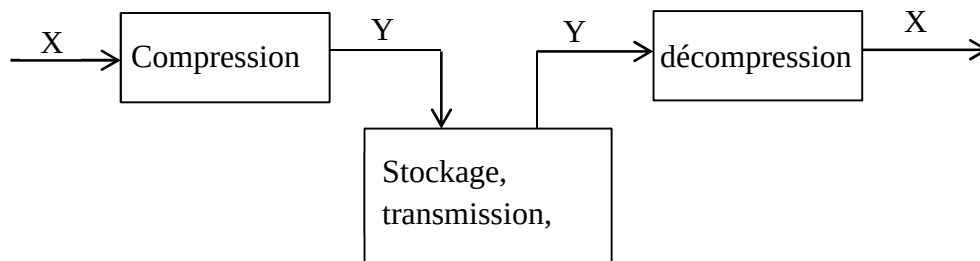


Figure I.1 : Schémas de compression sans perte.

4.1. Quelques domaines d'application

On trouve pas mal de domaine qui nécessite une compression sans perte, parmi ceux-là on cite :

- La grande majorité des logiciels ne sont pas conçu à être tolérants aux erreurs dans leurs fichiers. Si le logiciel détecte qu'un de ses fichiers est corrompu, il génère une erreur qui sera une obstruction pour son démarrage. Donc il est vraiment impératif d'utiliser une méthode de compression sans perte pour ce type de données.

- Dans un groupe de senseurs, en un temps précis, chaque senseur capte une image correspondant à un tranché, pour éviter un traitement erroné de groupe d'image due à la méthode de compression, on doit utiliser la compression sans perte.

4.2. Types d'algorithmes de compression sans perte

4.2.1. Méthode statistique [3]

Une technique basée sur les probabilités, le principe est d'associer le nombre d'occurrence à chaque mot de la source S, puis trouver un codage préfixe de la source qui tienne compte de la probabilité d'occurrence de chaque mot. Le nombre d'occurrence $f(s)$ peut être remplacé par la probabilité suivante : $p(s) = \frac{f(s)}{|S|}$

$p(s)$: probabilité d'apparition de s.

$f(s)$: fréquence d'apparition de s.

$|S|$: la somme des fréquences d'apparition de tous les caractères.

4.2.1.1. Codage de Huffman [3][5]

Le codage de Huffman est un type de compression basé sur la méthode statistique. Il a été mis en œuvre par David Huffman en 1952 au MIT (Massachusetts Institute of technology).

Le principe de codage de Huffman est d'observer la fréquence d'apparition de chaque symbole de texte à coder. Ensuite choisir les deux symboles les moins fréquents (deux symboles fils) et attribuer 0 pour le fils gauche et 1 pour celui de la droite en respectant l'ordre des fréquences, former un nouveau symbole père et sommer leurs comptes de fréquences et le mémoriser, puis Les deux fils sont retirés de la liste. Refaire cette action jusqu'à ce qu'il ne reste qu'un seul père représentant la racine de l'arbre.

Pour coder un symbole, il suffit de parcourir l'arbre à partir de la racine jusqu'au symbole.

Si la fréquence d'apparition est élevée le codage sera avec des chaînes de bits plus courtes, et avec des chaînes de bits longues sinon.

- **Exemple**

Soit à coder la chaîne AABCDCCCBB en appliquant le codage de Huffman

1) compléter la table de fréquence d'apparition de chaque symbole.

Symbole s	A	B	C	D
Fréquence d'apparition f(s)	2	2	3	3

Tableau I.1 : Tableau des fréquences d'apparition des symboles

2) construire l'arbre binaire comme suit :

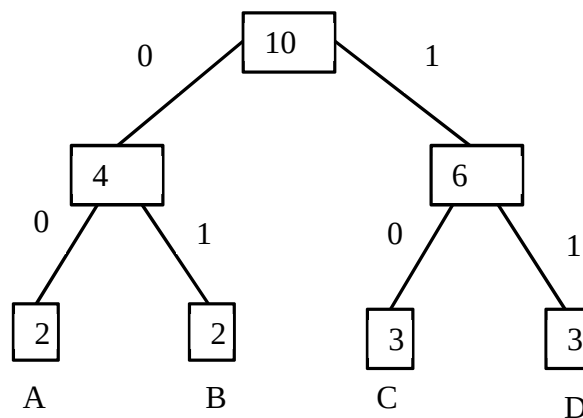


Figure I.2 : L'arbre construit selon la méthode de Huffman

3) Le codage de chaque symbole est :

A : 00

B : 01

C : 10

D : 11

Le codage de la chaîne est : 00 00 01 10 11 11 11 10 10 01

La longueur du texte codé vaut : $(2+2+3+3)*2=20$ bit

La longueur du texte original vaut : $(2+2+3+3)*8=80$ bit

Le taux de ce codage est : $T = (20/80)=0,25$.

Le gain de volume : $G=1-T=0,75$.

Pour le décodage, il suffit de reconstruire l'arbre et déduire le codage de chaque symbole.

Par exemple, pour trouver le caractère codé par 00 on se pointe sur la racine, on rencontre 0 donc on va à gauche, puis 0 alors on va aussi à gauche, enfin on rencontre une feuille qui est le caractère A. on refait cela pour tous les codes.

4.2.1.2. Codage de Shannon-Fano [3]

L'idée de ce codage est de répartir les symboles de texte à coder dans deux groupes de manière à ce que les valeurs des deux groupes soient à peu près équivalentes, donner **0** pour le groupe de gauche et **1** pour le groupe de droite. Chacun des groupes est subdivisé de cette manière jusqu'à ce qu'il ne reste qu'un seul symbole.

A la fin en aura un arbre binaire, où chaque branche représente un bit d'information et chaque feuille contient un symbole. Pour déduire le codage de chaque symbole, il suffit de parcourir l'arbre depuis la racine jusqu'au symbole.

- **Exemple**

Soit à coder la chaîne BANANES en codage Shannon-Fano

1) Créer la table des fréquences

s	A	N	B	E	S
f(s)	2	2	1	1	1

Tableau I.2 : Tableau des fréquences d'apparition des symboles

2) on construit l'arbre de la manière suivante :

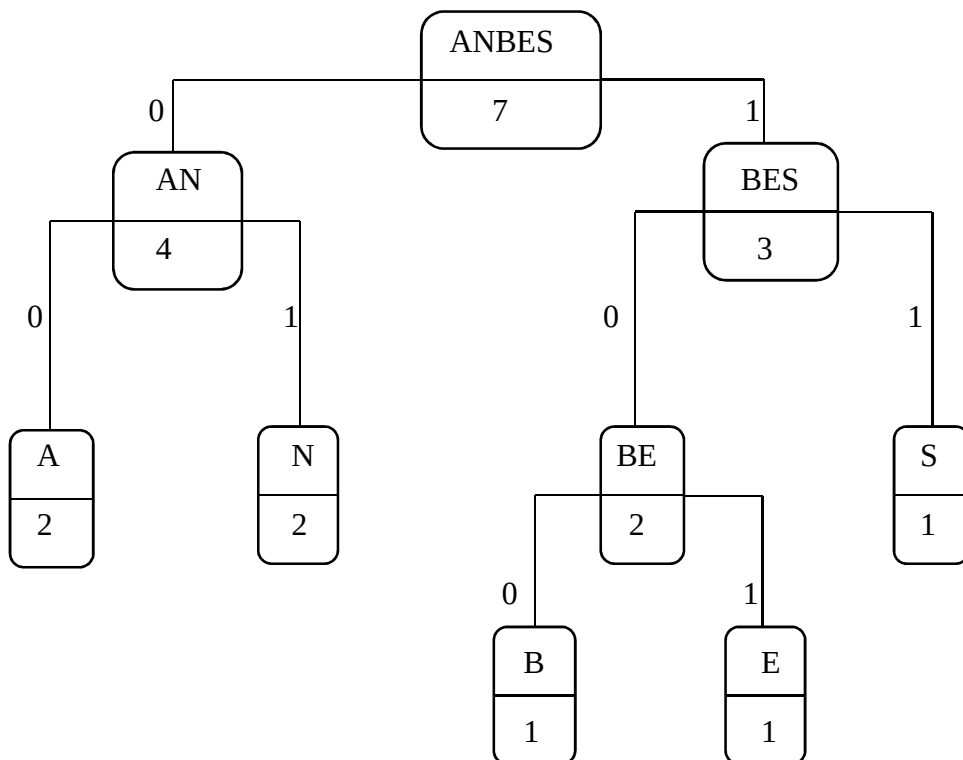


Figure I.3 : L'arbre construit selon la méthode de Shannon-Fano

3) le codage de chaque caractère est :

A : 00

N : 01

B : 100

E : 101

S : 11

Le codage de la chaîne est : 100 00 01 00 01 101 11

La longueur de texte codé est : $(2+2+1)*2 + (1+1)*3=16$ bit

La longueur de texte original est : $(2+2+1+1+1)*8=56$ bit

Le taux de ce codage est : $T = (16/56) = 0,029$

Le gain de volume est : $G = 1 - T = 0,71$

Lors de la décompression, il suffit juste de parcourir l'arbre à partir de la racine jusqu'aux feuilles et reconstruire le fichier original.

Par exemple, pour trouver le caractère codé par 100 on se pointe sur la racine, on rencontre 1 donc on va à droite, ensuite 0 alors on va à gauche, puis 0 alors on va aussi à gauche, enfin on rencontre une feuille qui est le caractère B. on refait cela pour tous les codes.

4.2.2. Méthodes de compression avec mémoire

4.2.2.1. Codage arithmétique [6]

Cette technique se repose sur la représentation de la suite de données comme un nombre réel entre 0 et 1. À mesure que la suite de données est plus longue, l'intervalle requis pour représenter la valeur diminue.

Le problème qui se pose dans le codage arithmétique, est le fait qu'on ne connaît pas à priori les fréquences exactes des symboles.

Parmi les compresseurs qui utilisent cette approche on cite :

- DMC (Dynamics Markov Coding) : ce codage débute avec un modèle d'ordre zéro et il s'étend graduellement au fur et à mesure que la compression progresse.
- PPM (Prediction by Partial Matching) : Ce codage cherche une coïncidence du texte dans un contexte d'ordre n . Si aucune correspondance (match) est trouvée, il descend à un contexte d'ordre $n-1$, jusqu'à ce qu'il arrive à zéro.

4.2.2.2. Codage à base d'un dictionnaire (compression substitutionnelle)

La base de cette compression est de remplacer une suite de données par une référence à une occurrence préalable à cette suite de données.

Deux chercheurs Israéliens, Abraham Lempel et Jacob Ziv mettent au point une méthode de compression à la fin des années soixante appelée LZ (Lempel-Ziv).

Parmi les variantes de LZ on peut citer LZ77, LZ78, LZW etc...

1) LZ77 (Lempel-Ziv 1977) [7]

Cette méthode utilise une fenêtre d'une taille de N caractères (N est la taille du texte à compresser) divisée en deux parties.

La deuxième partie représentant le tampon (sachant que c'est cette partie qui est la première à rencontrer le texte) est d'une taille de F caractères, et la première partie représentant le dictionnaire est d'une taille de D caractères ($N-F$).

Initialement, la fenêtre est située D caractères avant le début du texte, de façon à ce que le tampon de lecture soit entièrement positionné sur le texte, et que le dictionnaire n'y soit pas. Les D caractères sont alors des espaces.

A tout moment, l'algorithme va rechercher, dans les D premiers caractères de la fenêtre, le plus long facteur qui se répète au début du tampon de lecture. Il doit être de taille maximale F . Cette répétition sera alors codée par le triplet (i, j, c) :

- i est la distance entre le début du tampon et la position de la répétition dans le dictionnaire.
- j est la longueur de la répétition.
- c est le premier caractère du tampon différent du caractère correspondant dans le dictionnaire.

Après avoir codé cette répétition, la fenêtre coulisse de $j+1$ caractères vers la droite. Le codage du caractère c ayant provoqué la différence est indispensable dans le cas où aucune répétition n'est trouvée dans le dictionnaire. On codera alors $(0, 0, c)$.

Chaque chaîne qui disparaît de la fenêtre est supprimée du dictionnaire.

Cette méthode est utilisée dans les systèmes de fichier Windows NTFS.

- **Exemple**

Soit à coder la chaîne « **aacaacabcabaaac** » en appliquant LZ77 avec un dictionnaire de taille 6 et un tampon de pré-lecture de taille 4. La position du curseur est encadrée, le dictionnaire est en italique, et le tampon de pré-lecture est souligné.

Compression

Étape	Chaîne d'entrée														Code de sortie	
1	<u>a</u>	a	c	a	a	c	a	b	c	a	b	a	a	a	c	(0, 0, a)
2	a	<u>a</u>	c	a	a	c	a	b	c	a	b	a	a	a	c	(1, 1, c)
3	a	a	c	<u>a</u>	a	c	a	b	c	a	b	a	a	a	c	(3, 4, b)
4	a	a	c	a	a	c	a	b	<u>c</u>	<u>a</u>	<u>b</u>	<u>a</u>	a	a	c	(3, 3, a)
5	a	a	c	a	a	c	a	b	c	a	b	a	<u>a</u>	<u>a</u>	<u>c</u>	(1, 2, c)

Figure I.4: Exemple de compression LZ77.

La dernière étape ne trouve pas l'équivalence la plus longue (10, 3, (eof)) puisqu'elle est en dehors de la fenêtre.

Décompression

Étape	Chaîne d'entrée	Chaîne de sortie
1	(0, 0, a) <u>a</u>	→ a
2	(1, 1, c) a <u>a</u> c	→ a a c
3	(3, 4, b) a a c <u>a</u> a c a b	→ a a c a a c a b
4	(3, 3, a) a a c a a c a b <u>c</u> a b a	→ a a c a a c a b c a b a
5	(1, 2, c) ... a c a b c a b a <u>a</u> a c	→ ... a c a b c a b a a a c

Figure I.5: Exemple de décompression LZ77.

2) LZ78 (Lempel-Ziv 1978) [7] [8]

Cette technique est un peu plus facile que LZ77 (elle n'a pas de fenêtre glissante). Le dictionnaire est vide au départ et ne contient aucun facteur.

Au fur et à mesure de la compression, lorsque l'algorithme rencontre une chaîne de texte **chaîne1** composée d'un facteur (chaîne) **chaîne** qui existe dans le dictionnaire suivi d'un caractère **c** (sous la forme (**chaîne**, **c**)), il l'encode en attribuant le numéro **num** correspondant à **chaîne** suivi du caractère **c** sous la forme (**num**, **chaîne1**) sachant que **chaîne1**= (**chaîne**, **c**), puis il ajoute **chaîne1** au dictionnaire.

Au début du texte on ajoute un facteur ϵ (ϵ signifie le facteur vide) qui est numéroté par 0.

L'algorithme répète cette action jusqu'au dernier caractère du texte.

La taille du dictionnaire est limitée par le mode de codage (16,32 ou 64 bits).

- **Exemple**

Soit à coder la chaîne « **abaabbbbaabaaabbb** » en appliquant LZ78

chaîne	ϵ	a	b	aa	bb	ba	Ab	aaa	bbb
Num	0	1	2	3	4	5	6	7	8
codage		(0, a)	(0, b)	(1, a)	(2, b)	(2, a)	(1, b)	(3, a)	(4, b)

Tableau I.3 : Codage de la chaîne selon la méthode LZ78

Lors de la décompression, le décompresseur prend le flux de codes émis par le compresseur et l'utilise pour recréer le flux d'entrée exact. Une raison de l'efficacité de LZ78 est qu'il n'est pas nécessaire de passer le dictionnaire au décompresseur. Le dictionnaire peut être construit exactement comme il était pendant la compression, en utilisant le flux d'entrée comme données. Ceci est possible car le compresseur émet toujours un code avant de l'utiliser dans le flux de sortie, les données compressées ne sont donc pas surchargées par un gros dictionnaire."

3) LZW (Lempel-Ziv-Welch) [3][5]

Comme son nom l'indique, cette technique est une dérivée de LZ (plus précisément une modification de LZ78). Elle a été mise en œuvre à UNISYS par Terry Welch en 1984 pour implémenter des contrôleurs de disques à plus haute performances.

L'idée de Welch était d'améliorer LZ78, et cela en éliminant le caractère qui suit le numéro de chaîne, ce qui mène à coder qu'une seule information au lieu de deux pour LZ78.

En pratique, initialement le dictionnaire est rempli de 256 chaînes d'un caractère chacune (la table de code ASCII de 0 à 255). En parcourant le texte à coder du premier caractère jusqu'au dernier, lorsque l'algorithme rencontre une chaîne qui n'est pas au dictionnaire il l'ajoute à ce dernier. La première chaîne ajoutée au dictionnaire aura l'adresse 256.

Pour la décompression, Lors de la lecture de l'information encodée, les " mots " codés sont remplacés dans le fichier par leur correspondance lue dans le dictionnaire et le fichier d'origine est ainsi reconstitué.

Cette méthode est peu efficace pour les images, mais très recommandée pour les données texte et les données informatiques en général (plus de 50%).

• Exemple

Soit à coder la chaîne « SANANA-BANANA » en LZW.

Etape	Val s	Val t	Val u	u ∈ Dico ?	Ajouter u à Dico	Ecrire @ de
1	S	A	SA	Non	SA $\Rightarrow$ 256	Soit S 83
2	A	N	AN	Non	AN $\Rightarrow$ 257	Soit A 65
3	N	A	NA	Non	NA $\Rightarrow$ 258	Soit N 78
4	A	N	AN	Oui		
5	AN	A	ANA	Non	ANA $\Rightarrow$ 259	Soit AN 257
6	A	-	A-	Non	A- $\Rightarrow$ 260	Soit A 65
7	-	B	-B	Non	-B $\Rightarrow$ 261	Soit - 45
8	B	A	BA	Non	BA $\Rightarrow$ 262	Soit B 66
9	A	N	AN	Oui		
10	AN	A	ANA	Oui		
11	ANA	N	ANAN	Non	ANAN $\Rightarrow$ 263	Soit ANA 259
12	N	A	NA	Oui		
13	NA					Soit NA 258

Tableau I.4 : Tableau représentant la compression de la chaîne donnée selon la méthode LZW

S : symbole lu.
 T : symbole suivant.
 U : S+T.
 D : dictionnaire.

Le dictionnaire obtenu est le suivant :

Mot	Code
ASCII	0..255
SA	256
AN	257
NA	258
ANA	259
A-	260
-B	261
BA	262
ANAN	263

Tableau I.5 : Dictionnaire obtenu après compression LZW

Le codage sera : 83.65.78.257.65.45.66.259.258

La taille de fichier compressé est : $(9 \times 9) = 81$ bit

La taille de fichier original est : $13 \times 8 = 104$ bit

Le gain en espace est : $G = (1 - (81/104)) \times 100 = 22\%$

Pour décompresser le fichier, il suffit de chercher les codes dans le dictionnaire et reformuler le fichier original.

5. Compression avec perte (Lossy compression)

La compression avec perte (dite aussi compression destructive) est totalement différente de la compression sans perte. Son principe est de prendre en entrée une suite de bits X, choisir quelques bits à garder (les bits jugés significatifs) et le reste est détruit, ce qui donne une suite de bits Y. lors de la décompression, la suite de bits Z restituée est approximatives à la suite de bits originales X.

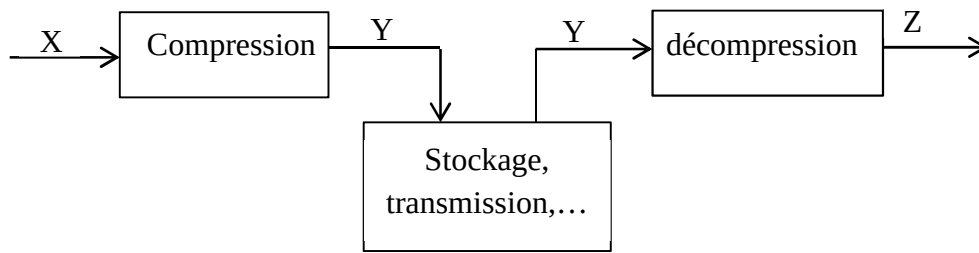


Figure I.6 : Schémas de compression sans perte.

Sachant que l'ouïe et la vue humaine ne sont pas parfaites, de nombreuses informations contenues dans les fichiers audio, image ou vidéo semblent inutile, car elles sont imperceptibles par nos sens.

En exploitant les faiblesses de l'homme, Ce type de compression est utilisé beaucoup plus sur les fichiers audio, image ou vidéo.

5.1. Compression d'image fixe

5.1.1. JPEG

La norme JPEG (Joint Photographic Expert Group) est la méthode de compression avec perte d'image fixe la plus répandue au monde, elle est définie par un groupe d'experts de la photographie en 1991.

On peut décomposer le processus de compression en cinq étapes :

5.1.1.1. Séparation de l'image en macro blocs

Vu la difficulté de faire un traitement sur l'image complète (temps de calcul et nombre de variables élevés), cette phase permet de découper l'image originale en bloc de 8x8 pixels (carrés de 64), où chaque bloc est traité séparément.

5.1.1.2. Transformation des couleurs

Cette étape exploite le manque de sensibilité de la vision humaine vis-à-vis les couleurs, en effectuant une conversion entre les systèmes RVB et YCrCb.

Y représente la luminance, Cr et Cb représentent la chrominance.

La conversion se fait selon les formules suivantes :

$$\left\{ \begin{array}{l} Y = 0,257 R + 0,504 G + 0,098 B + 16 \\ Cr = 0,439 R - 0,368 G - 0,071 B + 128 \\ Cb = -0,148 R - 0,291 G + 0,439 B + 128 \end{array} \right.$$

Et

$$\left\{ \begin{array}{l} R = 1,164 (Y - 16) + 1,596 (Cr - 128) \\ G = 1,164 (Y - 16) - 0,813 (Cr - 128) - 0,391 (Cb - 128) \\ B = 1,164 (Y - 16) + 2,018 (Cr - 128) \end{array} \right.$$

- **Exemple**



Image 8x8 pixel

52	55	61	66	70	61	64	73
63	59	55	90	109	85	69	72
62	59	68	113	144	104	66	73
63	58	71	122	154	106	70	69
67	61	68	104	126	88	68	70
79	65	60	70	77	68	58	75
85	71	64	59	55	61	65	83
87	79	69	68	65	76	78	94

*matrice de réel issue
d'une matrice de pixel*

Figure I.7 : Conversion d'une matrice de pixel en une matrice de réel

5.1.1.3. sous-échantillonnage de la chrominance

Après avoir transformé les couleurs, on est plus obligé de coder la chrominance pour chaque pixel. Mais plutôt la chrominance moyenne de carrés de 2x2 pixels, ce qui fait réduire l'espace les dimensions (horizontale et verticale) et diminuer le nombre de pixels à coder.

5.1.1.4. DCT

Cette phase consiste à passer d'une image NxN pixels à une matrice NxN de nombre réels.

L'équation de la transformée DCT s'exprime comme suit :

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

Avec :

- $C(z) = \begin{cases} \frac{1}{\sqrt{2}} & \text{pour } z = 0 \\ 1 & \text{sinon} \end{cases}$
- i, j les indices de la matrice de DCT.
- x, y les indices de l'image de départ.

L'équation inverse de la transformée DCT s'exprime comme suit :

$$pixel(x, y) = \frac{2}{N} \sum_{u=0}^{N-1} \sum_{v=0}^{N-1} C(u)C(v) \cos\left(\frac{(2x+1)u\pi}{2N}\right) \cos\left(\frac{(2y+1)v\pi}{2N}\right) DCT(u, v)$$

- **Exemple**

On soustrait à la matrice précédente 128 pour placer ses éléments dans l'intervalle

$[-2^{8-1} .. 2^{8-1} - 1]$, puis on applique DCT on aura :

$$\begin{bmatrix} -76 & -73 & -67 & -62 & -58 & -67 & -64 & -55 \\ -65 & -69 & -73 & -38 & -19 & -43 & -59 & -56 \\ -66 & -69 & -60 & -15 & 16 & -24 & -62 & -55 \\ -65 & -70 & -57 & -6 & 26 & -22 & -58 & -59 \\ -61 & -67 & -60 & -24 & -2 & -40 & -60 & -58 \\ -49 & -63 & -68 & -58 & -51 & -60 & -70 & -53 \\ -43 & -57 & -64 & -69 & -73 & -67 & -63 & -45 \\ -41 & -49 & -59 & -60 & -63 & -52 & -50 & -34 \end{bmatrix} \xrightarrow{\text{DCT}} \begin{bmatrix} -415 & -30 & -61 & 27 & 56 & -20 & -2 & 0 \\ 4 & -22 & -61 & 10 & 13 & -7 & -9 & 5 \\ -47 & 7 & 77 & -25 & -29 & 10 & 5 & -6 \\ -49 & 12 & 34 & -15 & -10 & 6 & 2 & 2 \\ 12 & -7 & -13 & -4 & -2 & 2 & -3 & 3 \\ -8 & 3 & 2 & -6 & -2 & 1 & 4 & 2 \\ -1 & 0 & 0 & -2 & -1 & -3 & 4 & -1 \\ 0 & 0 & -1 & -4 & -1 & 0 & 1 & 2 \end{bmatrix}$$

5.1.1.5. Quantification

La quantification est la phase où on perd vraiment de l'information, et aussi la phase qui fait diminuer la taille de l'image. Elle consiste à diviser la matrice obtenue dans DCT par

une matrice appelée matrice de quantification (coefficient de perte) pour éliminer les hautes fréquences qui représentent le bruit et les détails moins importants pour l'œil humain.

Chaque compresseur JPEG possède une ou plusieurs matrices de quantification prédéfinies.

- **Exemple**

Soit la matrice de quantification suivante :

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

On divise la matrice DCT par Q on aura

$$\begin{bmatrix} -26 & -3 & -6 & 2 & 2 & -1 & 0 & 0 \\ 0 & -2 & -4 & 1 & 1 & 0 & 0 & 0 \\ -3 & 1 & 5 & -1 & -1 & 0 & 0 & 0 \\ -4 & 1 & 2 & -1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

5.1.1.6. Codage entropique

Avant la sauvegarde de l'image, un codage entropique est appliqué sur l'image.

➤ **Codage entropique des séquences intermédiaires**

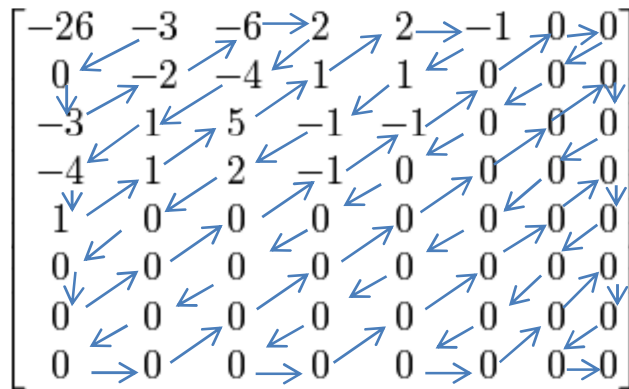
1) **Codage des coefficients DC**

Le premier coefficient DC de chaque bloc i varie peu d'un bloc à un autre. La séquence (DC^i) est codée en codage différentiel, ce qui veut dire coder la différence entre les deux coefficients DC successifs courant et précédents.

DC^0	$DC^1 - DC^0$	$DC^2 - DC^1$	$DC^3 - DC^2$	...	$DC^{p-2} - DC^{p-3}$
--------	---------------	---------------	---------------	-----	-----------------------

2) Codage des coefficients AC

Les coefficients AC de chaque bloc i sont lus en zigzag, formant une séquence globalement décroissante, dont une plage finale de coefficients nuls.



On aura

```

-3  0
-3 -2 -6
 2 -4  1 -4
 1  1  5  1  2
-1  1 -1  2  0  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  0  0
 0  0  0
 0  0
 0  0

```

➤ codage de Huffman

Un codage de Huffman est appliqué aux séquences pour améliorer le taux de compression et diminuer la taille sans perte.

Il suffit de faire la conversion YCrCb vers RVB pour avoir l'image compressée.

5.2. compression des signaux audio

➤ MP3

le principe est de calculer de façon dynamique la décomposition du signal à travers un ensemble de filtre, ensuite annuler les sons les moins forts (non audible) de façon à ce que l'oreille humaine ne fasse pas différence entre le fichier original et le fichier compressé en MP3 en se basant sur le modèle psycho-acoustique.

Cette technique permet un gain d'espace de 90% environ.

5.3. compression de vidéo

Vu la répétition de certaines images et le faible changement entre une image et une autre dans les séquences vidéos, les méthodes de compression de ce type de données reposent sur l'exploitation de ces redondances. Parmi les méthodes de compression de vidéo on cite le MPEG.

➤ MPEG (Motion Picture Experts Group)

La Méthode MPEG précède par l'analyse des changements d'une image par rapport à une autre image en étudiant les changements dans l'espace et les mouvements par bloc de pixels.

Certains blocs restent inchangés, d'autres subissent de légers déplacements et sont totalement modifiés. On déduit de la succession de plusieurs images un vecteur de mouvement par bloc.

A partir de ces éléments, on peut obtenir trois sortes d'images :

1. Image intra (image I) : sont des images qui servent de références aux autres images, elles sont compressées avec une variante de JPEG et leur taille est importante.
2. Image prédictive (image P) : sont des images codées en compression temporelle, à la fois avec les images déjà codées et avec prédiction de mouvements.

Elles indiquent seulement à quel autre bloc il ressemble dans l'image précédente et s'il y a un mouvement ou une erreur de prédiction.

Si le macrobloc ne change pas assez, il est ignoré. S'il change trop, il sera codé comme dans une image I.

3. Image bidirectionnelle (image B) : sont des images obtenues par interpolation à partir des images I et P situées avant et après en utilisant les vecteurs de mouvement.

6. conclusion

L'importance est surtout due au décalage qui existe entre les possibilités matérielles des dispositifs que nous utilisons (débits sur Internet et sur les divers câbles, capacités des mémoires de masse, ...) et les besoins qu'expriment les utilisateurs (visiophonie, vidéo plein écran, transfert de quantités d'information toujours plus importantes dans des délais toujours plus brefs). Quand ce décalage n'existe pas, ce qui est rare, la compression permet de toute façon de faire des économies.

La compression de données nous permettra toujours de résoudre des problèmes de volume, de stockage et de débit. De nombreux travaux de recherche sont en cours afin d'améliorer les performances des techniques de compression.

Chapitre II

La Méthode

LZW

1. Introduction

Contrairement aux méthodes statiques de compression de données qui emploient un modèle statistiques (dont la qualité de compression dépend du modèle), les méthodes de compression basé dictionnaire n'emploient ni un modèle statistique, ni des codes de tailles variables. Au lieu de cela, elles choisissent des cordes des symboles et codent chaque corde en tant que token en employant un dictionnaire.

Plusieurs algorithmes de compression basés sur cette méthode sont mis en œuvre, parmi ces algorithmes on peut citer en exemple LZ77, LZ78, LZW, LZS etc...

Dans notre cas on s'intéresse à l'algorithme LZW.

2. Définition de LZW [3]

La méthode LZW est l'une des meilleures méthodes de compression conservative, mais elle reste la moins utilisée en raison de sa protection par un brevet. Elle était la base des programmes de compression « zip » et les formats d'images « GIF » et « Tiff », et fait également partie du standard de compression de modem V.42bis et de PostScript niveau 2.

3. Historique de LZW [3]

En 1977, Abraham Lempel et Jacob Ziv ont introduit un algorithme de compression. Une année après (i.e. en 1978), ils modifièrent cet algorithme et le référencèrent par « compression basée dictionnaire ». L'abréviation du premier algorithme est LZ77 et son successeur est LZ78. Vu les problèmes de gestion de mémoire et de dictionnaire de la méthode LZ (LZ77 et LZ78), Terry Welch a modifié le LZ78 en 1984 en proposant un nouvel algorithme appelé LZW.

4. Caractéristiques de LZW [5]

La méthode LZW a certaines caractéristiques, comme :

- L'utilisation d'un dictionnaire qu'elle construit dynamiquement, au cours de la compression et de la décompression.
- Le dictionnaire n'est pas stocké dans le fichier compressé (pas de table d'entête).
- Elle comprime en une seule lecture (à la volé) ce qui donne une compression en un temps meilleurs que les méthodes statistique.

- Pour être efficace et reconnaître des longues chaînes, elle a besoin d'un apprentissage.

5. Principe général [5]

Dû au fait qu'une séquence de caractères peut apparaître plusieurs fois dans un même fichier, la compression avec LZW consiste à éviter les répétitions et économiser de la place. Le dictionnaire contiendra toutes les répétitions, et doit être créé à la volée et de la même manière dans les deux opérations (compression et décompression).

5.1. Compression

Dans le processus de compression, l'algorithme construit un dictionnaire où chaque séquence sera numérotée par une adresse dans celui-ci. Chaque séquence ne se trouvant pas dans le dictionnaire sera rajoutée. À la fin, on aura des adresses pointant vers une séquence contenue dans le dictionnaire. La taille de dictionnaire n'est pas fixe et les 256 premières adresses (de 0 à 255) du dictionnaire contiennent les codes ASCII. La première séquence formée aura l'adresse 256.

5.1.1. Algorithme de compression [5]

Données :

- le dictionnaire des symboles rencontrés : Dico
- le fichier à compresser : Fichier

Résultat :

- le fichier compressé : Fichier

L'algorithme :

début

s=premier octet du fichier

tant que le fichier n'est pas à sa fin

t=octet suivant

u=concaténation(s,t)

si (u appartient Dico)

s=u

sinon

ajouter (u) dans Dico

écrire adresse de s

s=t

fin si

fin tant que

écrire adresse de s

fin.

5.1.2. Exemple

Soit à compresser la chaîne « SANANA-BANANA » en LZW.

Explication de déroulement de l'algorithme

On lit le premier caractère soit 'S'. puis on lit le suivant 'A'. on forme la séquence $u=s+t$ soit 'SA', cette séquence n'est pas dans le dictionnaire, donc on l'ajoute dans ce dernier. cette séquence aura l'adresse 256. Enfin on attribue l'adresse de s (valeur ASCII de 'S') dans le fichier soit **83** (1010011 en binaire). On refait cette action jusqu'à la fin du fichier.

La compression effective aura lieu lorsqu'on rencontrera pour la seconde fois une paire ('AN' par exemple) déjà présenté dans le dictionnaire. Dans ce cas on émettra l'adresse de la séquence 'AN' et non pas l'adresse de la séquence 'A' et celle de la séquence 'N'. On ajoutera par la suite dans le dictionnaire la séquence de 3 caractères (soit 'ANA' dans notre exemple).

Au début les codes sont sur 8 bits pour les 256 premières entrées du dictionnaire (de 0 à 255), puis continue avec 9 bit pour les entrées de 256 à 511, et ainsi de suite.

Les étapes du déroulement de l'algorithme sont présentées dans le tableau ci-dessous :

Etape	Val s	Val t	Val u	$u \in \text{Dico} ?$	Ajouter u à Dico	Ecrire @ de
1	S	A	SA	Non	SA $\Rightarrow$ 256	Soit S 83
2	A	N	AN	Non	AN $\Rightarrow$ 257	Soit A 65
3	N	A	NA	Non	NA $\Rightarrow$ 258	Soit N 78
4	A	N	AN	Oui		
5	AN	A	ANA	Non	ANA $\Rightarrow$ 259	Soit AN 257
6	A	-	A-	Non	A- $\Rightarrow$ 260	Soit A 65
7	-	B	-B	Non	-B $\Rightarrow$ 261	Soit - 45
8	B	A	BA	Non	BA $\Rightarrow$ 262	Soit B 66
9	A	N	AN	Oui		
10	AN	A	ANA	Oui		
11	ANA	N	ANAN	Non	ANAN $\Rightarrow$ 263	Soit ANA 259
12	N	A	NA	Oui		
13	NA					Soit NA 258

Tableau II.1 : Tableau représentant le déroulement de la compression de la chaîne donnée selon la méthode LZW

s : symbole lu.
t : symbole suivant.
u : s+t.
Dico : dictionnaire.

Le dictionnaire résultant de l'algorithme est :

Mot	Code
ASCII	0..255
SA	256
AN	257
NA	258
ANA	259
A-	260
-B	261
BA	262
ANAN	263

Tableau II.2 : Dictionnaire obtenu après compression LZW

La chaîne compressée est : 83.65.78.257.65.45.66.259.258

5.2. décompression

Concernant la décompression, l'algorithme LZW reconstruit le dictionnaire au fur et à mesure de la lecture du fichier compressé. Ce processus est très proche de celui de la compression.

Le processus de compression débute par la lecture des codes du fichier en partant d'une taille de 8 bits. Puis suivant la méthode utilisée, lorsque le décompresseur rencontre un code spécial il augmente la taille de son tampon pour lire les adresses. Il continue à lire les adresses jusqu'à la fin du fichier et il inscrit la séquence correspondant à l'adresse dans le fichier décompresser.

5.2.1. Algorithme de décompression [5]

Données :

- le dictionnaire des symboles rencontrés : Dico
- le fichier à compresser : Fichier

Résultat :

- le fichier compressé : Fichier

L'algorithme :

début

a=Séquence(première adresse contenu dans le fichier)

écrire a

tant que le fichier n'est pas à sa fin

b=adresse suivante

si (b appartient Dico)

s=Séquence(b)

sinon

s=concaténation(a,t)

fin si

écrire séquence(s)

t=s[0]

ajouter (Séquence(a)+t) dans Dico

a=b

fin tant que

fin

5.2.2. Exemple

Soit à décompresser le « 83.65.78.257.65.45.66.259.258 ».

Explication de déroulement de l'algorithme

On lit la première adresse soit 83. On écrit la séquence associée à cette adresse du dictionnaire. Ensuite on lit l'adresse suivante. On s'assure que l'adresse lue appartient au dictionnaire, puis on écrit la séquence associée à cette adresse. On ajoute dans le dictionnaire la concaténation de l'ancienne séquence et la première lettre de la séquence en cours. On refait ensuite cette action jusqu'à la fin du fichier.

Les étapes du déroulement de l'algorithme sont présentées dans le tableau ci-dessous :

Etape	Val a	Val b	b ∈ Dico ?	Val s	Val t	Ajouter au Dico	Ecrire la séquence
1	S=séq(83)	65	Oui	A=séq(65)	A	SA $\Rightarrow$ 256	S puis A
2	A=séq(65)	78	Oui	N=séq(78)	N	AN $\Rightarrow$ 257	N
3	N=séq(78)	257	Oui	AN=séq(257)	A	NA $\Rightarrow$ 258	AN
4	AN=séq(257)	65	Oui	A=séq(65)	A	ANA $\Rightarrow$ 259	A
5	A=séq(65)	45	Oui	- =séq(45)	-	A- $\Rightarrow$ 260	-
6	- =séq(45)	66	Oui	B=séq(66)	B	-B $\Rightarrow$ 261	B
7	B=séq(66)	259	Oui	ANA=séq(259)	A	BA $\Rightarrow$ 262	ANA
8	ANA=séq(259)	258	Oui	NA=séq(258)	N	ANAN $\Rightarrow$ 263	NA

Tableau II.3 : Tableau représentant le déroulement de la décompression de la chaîne donnée selon la méthode LZW

Le dictionnaire résultant est le même que celui de la compression.

Mot	Code
ASCII	0..255
SA	256
AN	257
NA	258
ANA	259
A-	260
-B	261
BA	262
ANAN	263

Tableau II.4 : Dictionnaire obtenu après décompression LZW

On remarque que l'algorithme de décompression est plus rapide que celui de la compression (8 étapes contre 13 pour la compression dans notre exemple).

La chaîne décompressée est : « SANANA-BANANA ».

6. Les variantes de LZW

Il existe plusieurs variantes des compressions LZW, parmi celle-ci on trouve :

6.1. LZC [9]

LZC est la variante de l'algorithme LZW qui est utilisé dans le système d'exploitation libre UNIX. Il compresse selon le principe LZW. La longueur maximale d'un index peut être définie par l'utilisateur du programme de compression en tenant compte de la mémoire disponible (de 9 à 16 bits). Il commence d'abord par 9 bits pour les 512 premières entrées du dictionnaire, puis continue avec 10 bit pour les entrées de 512 à 1023, et ainsi de suite jusqu'à la limite spécifiées par l'utilisateur. Enfin, l'algorithme devient un dictionnaire statique, vérifie régulièrement le taux de compression. Quand il détecte une baisse de taux de compression, il jette le dictionnaire et commence à construire un nouveau à partir de zéro.

6.2. LZT

C'est une autre variation sur le thème LZW, l'algorithme est presque la variante LZC, la seule différence étant qu'il fait de la place pour de nouvelles phrases à ajouter au dictionnaire en enlevant le moins récemment utilisé.

7. Performance de LZW

On se basant sur l'exemple traité dans la section 5.1.2 on a :

La taille de fichier original est : $(13 \times 8) = 104$ bit

La taille de fichier compressé est : $(6 \times 8) + (3 \times 9) = 75$ bit

Le gain en espace est : $G = (1 - (75/104)) \times 100 = 28\%$

Lorsque la taille du fichier est grande, le taux de compression est supérieur à celui obtenu par une méthode de compression statistique. En plus de taux de compression, l'algorithme LZW est nettement plus rapide qu'un algorithme de type statistique en termes de traitement.

8. Conclusion

Avant l'année 2004, LZW était peu utilisée car elle était brevetée par la société Unisys. Mais après le 8 juillet 2004 où le brevet a expiré, Unisys ne peut plus réclamer les droits de son utilisation.

Aujourd'hui, l'algorithme LZW est considéré comme l'une des méthodes de compression les plus efficaces. Elle est relativement rapide ce qui a rendu l'utilisation de la compression possible sur les disques durs de façon transparente. Cette méthode est aussi utilisée dans le format .gif, est surtout dans la compression des paquets sur les réseaux. Mais encore dans les compresseurs tels que ZIP, ARJ.

Chapitre III

La Méthode

VLC

1. Introduction

Dans la plus part des fichiers, on constate que certains caractères se répètent beaucoup plus que d'autres. Pour cela, une technique très pratique qui tient compte des occurrences d'apparition des symboles a été mise en œuvre. Cette technique est la technique de codage à longueur variable. Plusieurs méthodes de codage ont adaptées cette technique comme les méthodes de : Huffman, Shannon etc...

2. Définition

2.1. Code à longueur variable (VLC) [15]

L'idée générale de codage à longueur variable (variable-length code) est d'associer des mots de codes plus courts aux symboles plus fréquents, et les mots de code plus longs aux symboles moins fréquents.

L'espérance de la longueur que l'on cherche à minimiser est donnée par :

$$L = \sum_{x_i \in \mathcal{A}} \Pr(X = x_i) l_i$$

Tel que : l_i est la longueur de mot de code associé au symbole x_i .

2.2. Code uniquement décodable [1]

Le but de la compression sans perte est de comprimer les données de telle façon qu'on puisse les reconstruire parfaitement (sans erreur).

Pour y faire, le décodage doit avoir lieu sans ambiguïté. Ceci dit qu'une séquence codée doit être interprétable d'une façon unique comme une succession de mots de codes déterminés.

Définition : Un code est uniquement décodable s'il peut être clairement décodé.

Supposons qu'on a de séquence de mots de code concaténés identiques :

$$c_1 c_2 \dots c_k = c'_1 c'_2 \dots c'_k$$

Où les c_i et c'_i appartient au code $\mathcal{C}$, alors les mots sont un à un identiques :

$$c_i = c'_i, \quad i=1 \dots k$$

Analysons les exemples de codes suivants :

X_i	Code 1	Code 2	Code 3	Code 4
0	0	0	10	0
1	0	10	00	10
2	1	100	11	110
3	1	101	110	111

- Code 1 : Un code uniquement décodable doit être inversible, i.e. les mots de code doivent être distincts. Ce qui ne s'applique au code 1, et cela signifie que ce code est ambigu. Alors il n'est pas utilisable.
- Code 2 : Ce code est inversible, mais pas uniquement décodable.

Par exemple la séquence codée $c = 101010\dots$ peut se décoder en 301 ou 130, alors ce code est inutilisable.

- Code 3 : Le troisième code est plus intéressant : malgré les apparences, il est uniquement décodable. En effet, on peut imaginer un décodeur qui examine la parité du nombre de zéros qui suit la première séquence « 11 » ; ce n'est qu'après avoir relu un « 1 » qu'il peut correctement découper la séquence codée et fournir les symboles de source. Cet exemple montre que certains codes uniquement décodables peuvent nécessiter une implantation complexe du décodeur, qui doit lire la séquence codée binaire suffisamment loin à l'avance pour décoder un symbole de source.

2.3. Code préfixe et arbre binaire

2.3.1. Code préfixe [2]

Un codage d'un ensemble de mots S est dit préfixe, si et seulement si aucun code d'un mot de S n'est préfixe d'un autre code d'un mot du même ensemble. autrement dit, aucun code ne débute un autre code.

2.3.2. Construction du l'arbre binaire [2]

La construction de l'arbre binaire se fait selon les règles suivantes :

1. La branche gauche est étiquetée par 0 et celle de droite par 1.
2. Chaque feuille est étiquetée par un mot de S .
3. Le code correspondant à un mot de S est le chemin parcouru depuis la racine jusqu'à la feuille.

• **Exemple**

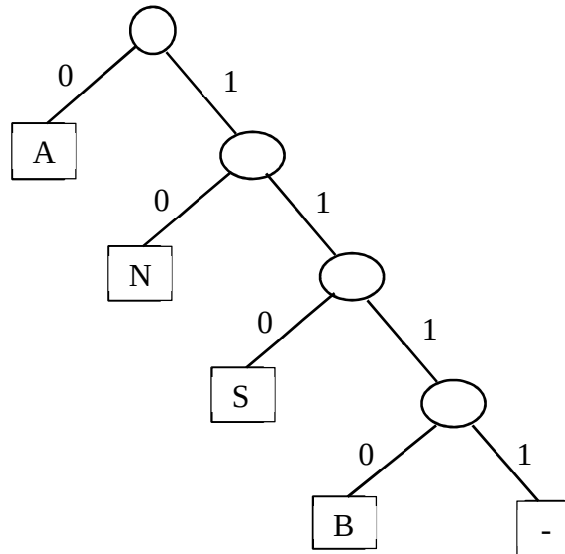


Figure III.1: Exemple d'arbre binaire

On note que :

- Pour tout code préfixe, il existe un arbre de codage le représentant.
- Le code construit par un arbre de codage est préfixe.

3. Inégalité de kraft-McMillan [13]

Cette inégalité est liée aux codes non ambigus de taille variable. Sa première partie est de déclarer un code de taille variable non ambigu, avec n codes de tailles L_i . Dans sa deuxième partie qui est l'inverse, étant donnée un ensemble de n entiers positifs $(L_1, \dots, L_n)$ satisfaisants l'équation ci-dessus, il existe un code de taille variable non ambigu tels que les L_i sont les tailles de ses différents codes.

$$\sum_{i=1}^n 2^{-L_i} \leq 1$$

Les deux parties ensemble indiquent qu'un code est non ambigu si et seulement s'il satisfait cette équation.

Cette inégalité peut être liée à l'entropie en voyant que les longueurs L_i peuvent toujours s'écrire comme suit : $L_i = \log_2 P_i + E_i$

Où E_i est la quantité par laquelle L_i est plus grand que l'entropie (la longueur supplémentaire de code i).

Ceci implique : $2^{-L_i} = 2^{(\log_2 P_i - E_i)} = 2^{\log_2 P_i} / 2^{E_i} = P_i / 2^{E_i}$

Dans le cas spécial où toutes les longueurs supplémentaire sont identiques ($E_i = E$), l'inégalité de Kraft mène à déduire que :

$$1 \geq \sum_{i=1}^n P_i / 2^{E_i} = 1 / 2^E \sum_{i=1}^n P_i = 1 / 2^E \Rightarrow 2^E \geq 1 \Rightarrow E \geq 0$$

Un code non ambigu a la longueur supplémentaire non négative, signifiant que sa longueur est supérieure à la longueur déterminée par son entropie.

4. Types de codage entropique

4.1. Codage statistique [2] [3]

Le codage statistique est basé sur les probabilités, ce type de codage remonte aux années quarante lorsque Claude Shannon avait démontré l'existence d'une méthode permettant de compacter les flux d'information sans rien prendre de leurs signification.

Le principe de ce type de codage est d'associer le nombre d'occurrence à chaque mot de la source S, puis trouver un codage préfixe de la source qui tienne compte de la probabilité d'occurrence de chaque mot. Parmi les algorithmes de codage statistique on trouve le codage de Shannon-Fano, codage de Huffman....

4.1.1. Codage de Shannon-Fano [3]

En même temps que Shannon, R.F Fano poursuivait au MIT (Massachusetts Institute of Technology) des recherches similaires. Dans la méthode connue sous le nom de "Shannon-Fano", les étapes de cette méthode sont comme suit :

- 1) Placer les fréquences d'apparition de chaque symbole dans un tableau suivant un ordre décroissant.
- 2) Diviser le tableau en deux groupes tels que la somme des fréquences dans chaque groupe (S_0 ou S_1) avoisine la somme totale des fréquences ($|S|$).
- 3) Attribuer à chaque sous ensemble un symbole binaire d'encodage distinct (0 pour S_0 et 1 pour S_1).

- 4) Si un sous ensemble S_i n'a qu'un seul élément c'est une feuille terminale, sinon répéter à partir de 2) pour chaque sous ensemble jusqu'à ce que chaque signe possède un code binaire distinct.

• **Exemple pratique**

Soit à encoder la chaîne « SANANA-BANANA » avec la méthode de Shannon-Fano.

➤ Etape 1 :

symbole	A	N	S	B	-
fréquence	6	4	1	1	1

Tableau III.1 : Table des fréquences des symboles

➤ Application successive des étapes 2, 3 et 4 :

- Première subdivision :

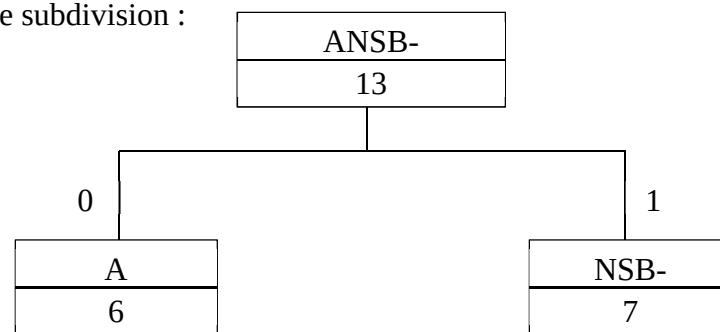


Figure III.2.a: Première subdivision pour la construction de l'arbre

- Deuxième subdivision :

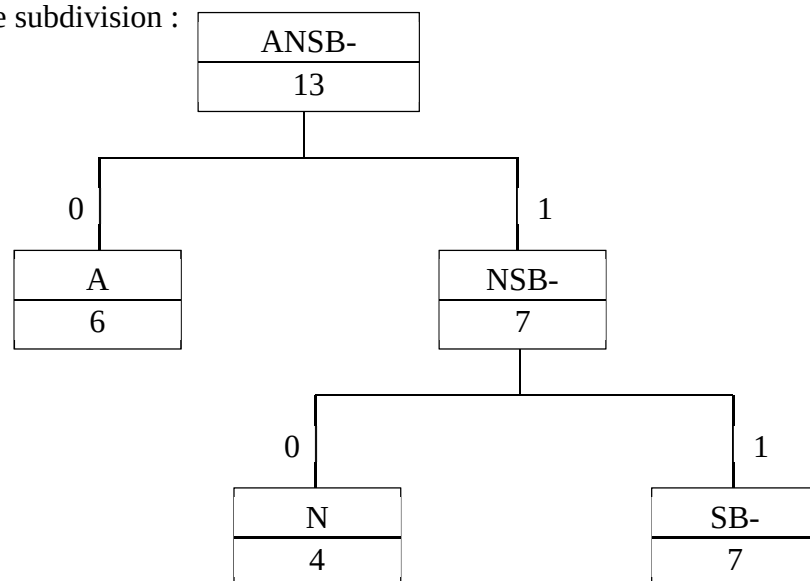


Figure III.2.b: Deuxième subdivision pour la construction de l'arbre

- Troisième subdivision :

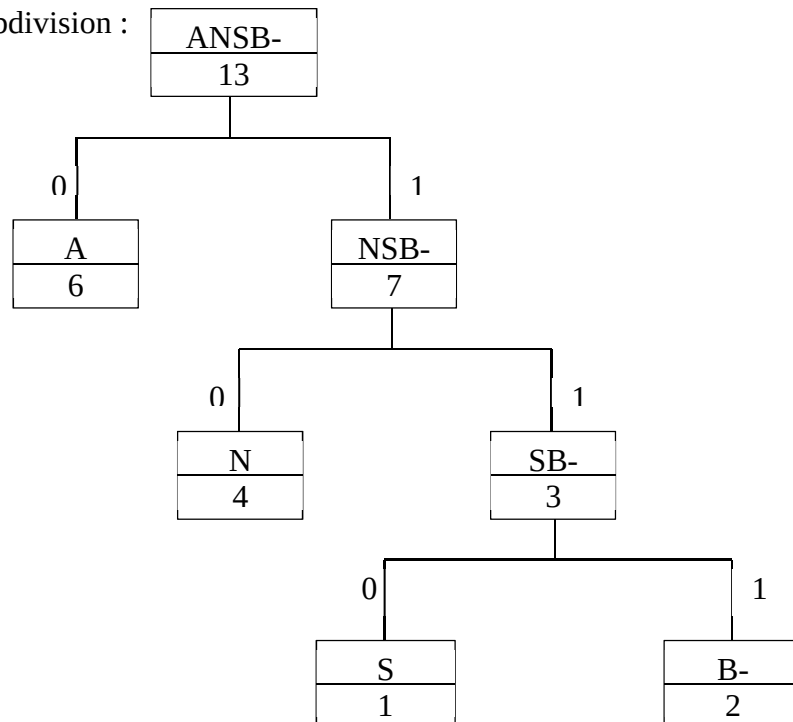


Figure III.2.c: Troisième subdivision pour la construction du l'arbre

- Quatrième subdivision :

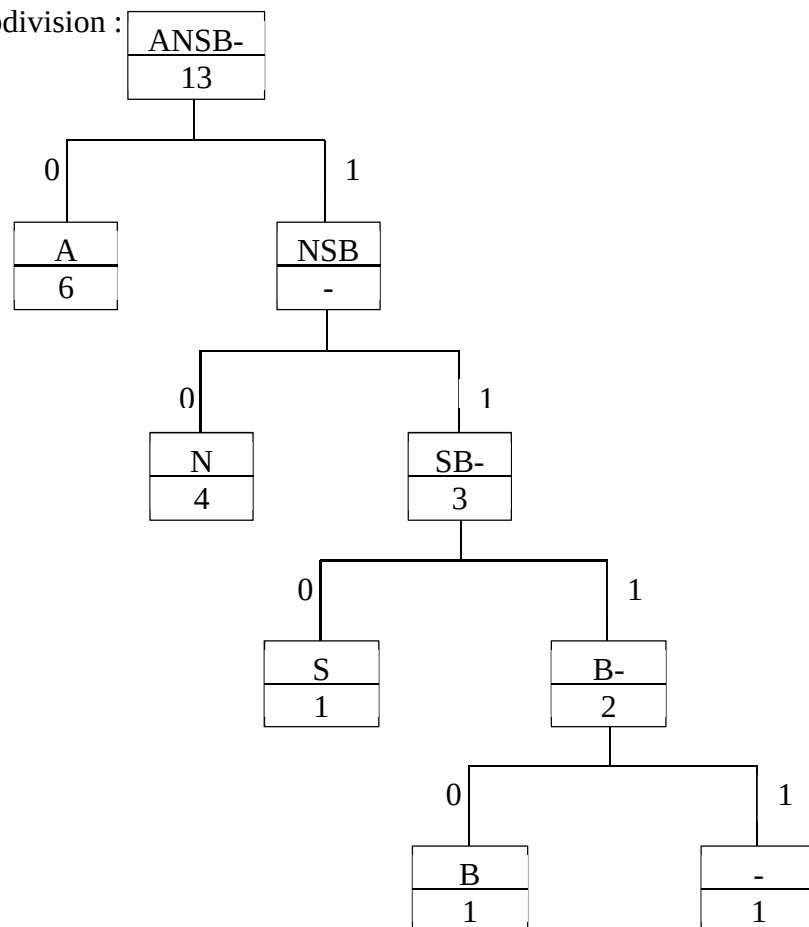


Figure III.2.d: Quatrième subdivision pour la construction du l'arbre

A partir de cet arbre on a la table des codes suivants :

Symbole	code
A	0
N	10
S	110
B	1110
-	1111

Tableau III.2 : Table des codes des symboles

La chaîne codée est : « 110 0 10 0 10 0 1111 1110 0 10 0 10 0 »

Propriété [2]

- L'algorithme de Shannon-Fano construit un codage préfixe.
- l'arbre de Shannon-Fano n'est pas toujours optimal.

4.1.2. Codage de Huffman [3] [5]

Le codage de Huffman est le résultat des recherches de David Huffman, mis en œuvre en 1952. L'étude de Huffman repose sur le fait que certains caractères sont plus répétés que d'autres dans un même fichier.

Le principe de ce codage est

- 1) Placer les fréquences d'apparition de chaque symbole dans un tableau suivant un ordre décroissant.
- 2) choisir les deux symboles ou arbres les moins fréquents et attribuer le code 0 pour celui de gauche et le code 1 pour celui de droite.
- 3) Regrouper les deux symboles en formant un arbre binaire et attribuer à cet arbre une fréquence d'apparition qui est la somme des fréquences d'apparition de ses deux fils.
- 4) Si la fréquence d'apparition obtenue est inférieure à la fréquence d'apparition totale alors on refait les étapes à partir de 2).

• Exemple pratique

Soit à encoder la chaîne « SANANA-BANANA » avec la méthode de Huffman.

➤ Etape 1 :

symbole	A	N	S	B	-
fréquence	6	4	1	1	1

Tableau III.3 : Table des fréquences des symboles

➤ Application successive des étapes 2, 3 et 4 :

▪ Première répartition :

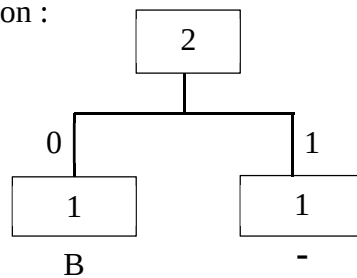


Figure III.3.a: Première répartition pour la construction du l'arbre

▪ Deuxième répartition :

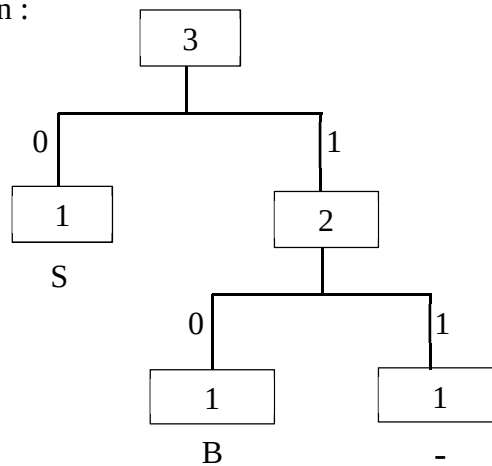


Figure III.3.b: Deuxième répartition pour la construction du l'arbre

- Troisième répartition :

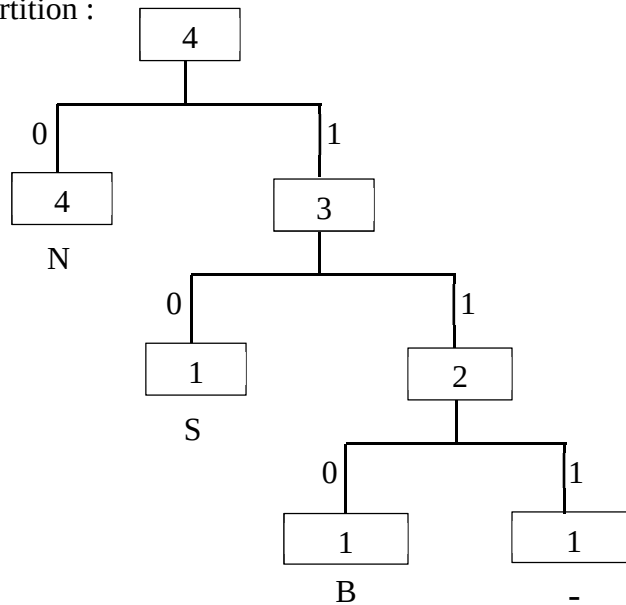


Figure III.3.c: Troisième répartition pour la construction du l'arbre

- Quatrième répartition :

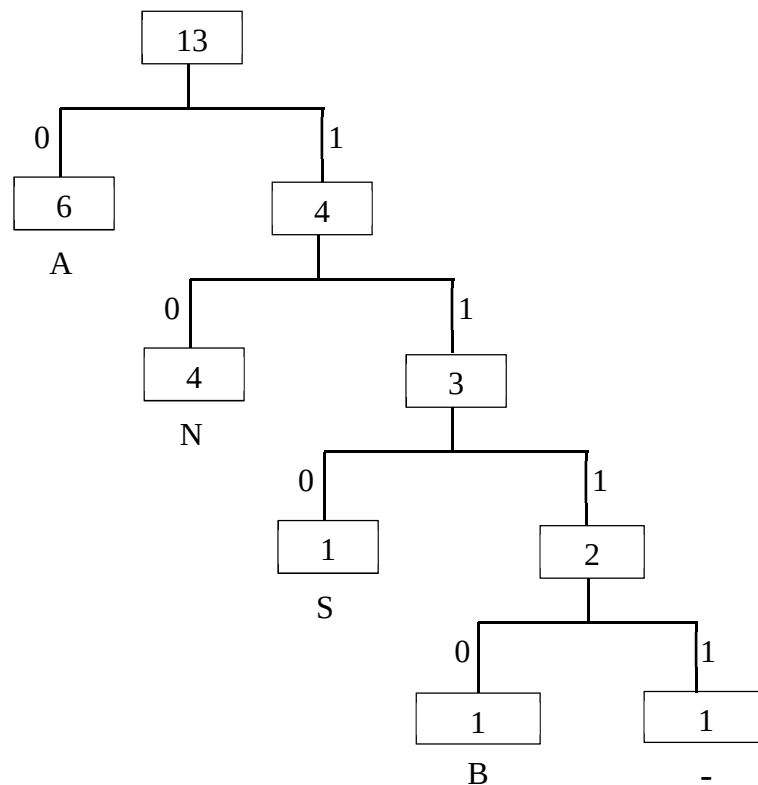


Figure III.3.d: Quatrième répartition pour la construction du l'arbre (l'arbre final)

A partir de cet arbre on a la table des codes suivantes :

symbole	code
A	0
N	10
S	110
B	1110
-	1111

Tableau III.4: Code correspondant aux caractères codés

La chaîne codée est : « 110 0 10 0 10 0 1111 1110 0 10 0 10 0 »

Propriété [2]

- L'algorithme de Huffman construit un codage préfix.
- l'arbre de Huffman est optimal.

4.2. Le code unaire (unary code) [4]

Le code unaire des nombres entiers est le code à longueur variable le plus simple.

Le code unaire d'un nombre n est construit d'une suite n **1** suivi d'un **0** (ou n **0** suivi d'un **1**) comme montré dans le tableau suivant :

N	Code	Code inverse
0	0	1
1	10	01
2	110	001
3	1110	0001
...	...	...
I	111...10	000...01

Tableau III.5: Exemple de code unaire

L'inconvénient de ce type de codage est que sa taille est d'ordre exponentiel. C'est-à-dire pour coder les caractères de la table ascii, le premier caractère sera codé sur 1 bit et le dernier qui est le 256^{ème} sera codé sur 256 bits (32 octets).

4.3. Codage arithmétique [6]

Le codage arithmétique est un codage utilisant un modèle statistique, tout comme le codeur de Huffman. Contrairement à ce dernier, il produit un code pour une séquence de symboles tout entière, et non pas un code par symbole. Chaque nouveau symbole lu modifie de façon incrémentale le code de sortie. Ce code de sortie est un nombre à virgule flottante compris entre 0 et 1, dont le nombre de chiffres après la virgule correspond au nombre de symboles. Contrairement à Huffman, il n'est pas obligatoire que chaque code ait un nombre entier de bits. Par exemple, un symbole de probabilité 0.9 a pour entropie 0.15, mais Huffman affectera probablement un code de un bit (ou plus), et la séquence codée aura un nombre de bits plus long qu'en théorie. Le codeur arithmétique est plus performant que le codeur de Huffman, mais il est plus complexe à implémenter.

4.3.1. Codes d'Elías [10] [11]

C'est un encodeur arithmétique, il calcule le code $C(x^t)$ qui correspond à la séquence source x^t . le décodeur correspondant reconstruit la séquence source à partir de code en exécutant des calculs simples.

La première idée derrière le codage d'Elías est que à toute séquence source x^t correspond un sous-intervalle $[0..1[$. Autrement dit, la distribution de probabilité des symboles de la source vérifie la relation suivante :

$$\sum_{i=1}^M p_i = 1$$

Nous pouvons donc construire un découpage du segment $[0..1[$ en intervalles contigus $I_1, I_2, \dots, I_M$, où chaque I_i est de longueur p_i . dans la suite, chaque point du segment $[0..1[$ est un nombre qui sera représenté par son développement binaire.

Trois principaux algorithmes ont été proposés par Peter Elias.

4.3.1.1. Code Gamma $\gamma(N)$ [12]

Le code Gamma $\gamma(N)$ est l'un des codes proposés par Peter Elias pour les nombres entiers positifs, il est simple pour coder et décoder et est également universel.

➤ Codage

On donne un nombre entier positif N et on exécute les étapes suivantes :

1. Trouver le plus grand nombre entier M tel que $2^M \leq N \leq 2^{M+1}$ et écrire $N = 2^M + L$.

Avec L est au plus M bits.

2. Coder le nombre M en code unaire sous la forme suivante :

$$\left\{ \begin{array}{ll} M \ll 0 \gg \text{ suivis de } 1 & \underbrace{00\dots\dots 01}_{M \text{ fois}} \\ \text{Ou} & \\ M \ll 1 \gg \text{ suivi de } 0 & \underbrace{11\dots\dots 10}_{M \text{ fois}} \end{array} \right.$$

3. Concaténer $U(M)$ et $B(L)$ tel que

$$\left\{ \begin{array}{l} U(M) \text{ est le code unaire de } M \\ B(L) \text{ est la représentation de } L \text{ en base 2 sur } M \text{ bit.} \end{array} \right.$$

• Exemple

Les codes des 20 premiers entiers (à partir de 1) sont représentés dans le tableau suivant :

Nombre $N=2^M + L$	Code $U(M)B(L)$	Nombre	Code
$1 = 2^0 + 0$	1	$11 = 2^3 + 3$	0001011
$2 = 2^1 + 0$	010	$12 = 2^3 + 4$	0001100
$3 = 2^1 + 1$	011	$13 = 2^3 + 5$	0001101
$4 = 2^2 + 0$	00100	$14 = 2^3 + 6$	0001110
$5 = 2^2 + 1$	00101	$15 = 2^3 + 7$	0001111
$6 = 2^2 + 2$	00110	$16 = 2^4 + 0$	000010000
$7 = 2^2 + 3$	00111	$17 = 2^4 + 1$	000010001
$8 = 2^3 + 0$	0001000	$18 = 2^4 + 2$	000010010
$9 = 2^3 + 1$	0001001	$19 = 2^4 + 3$	000010011
$10 = 2^3 + 2$	0001010	$20 = 2^4 + 4$	000010100

Tableau III.6: Codage Gamma des 20 premiers entiers positifs

➤ Décodage

Pour le décodage on suit les étapes suivantes :

1. Lire les zéros du code jusqu'à ce qu'un 1 soit produit. Dénoter les zéros, et leurs nombre représente M.
2. Lire les M bits qui suivent comme un entier L. et calculer $N = 2^M + L$.

Le code Gamma peut être employé pour coder des nombres entiers positifs même dans les cas où le plus grand nombre entier n'est pas connu d'avance.

4.3.1.2. Code Delta $\delta(N)$ [12]

Déferrement de son code Gamma où il ajoute la longueur au début de code (dans le code unaire), un autre code pour la construction d'un code VLC à était proposé par Elias, c'est le code Delta $\delta(N)$. Dans ce dernier, Elias ajoute la longueur dans le code binaire.

➤ Codage

Le processus du codage Delta d'Elias ce peut se faire selon deux méthodes :

a. Première méthode

Cette méthode utilise le codage Gamma d'Elias, et se déroule comme suit :

1. Trouver le plus grand nombre entier M tel que $2^M \leq N \leq 2^{M+1}$ et écrire $N = 2^M + L$. Avec L est au plus M bits.
2. Coder M+1 avec le code Gamma.
3. Concaténer les deux codes $\gamma(M+1)$ et B(L).

• Exemple

Soit à coder le nombre 20 en utilisant le première méthode de Delta

$$N = 20 = 2^4 + 4$$

$$\gamma(M+1) = \gamma(4+1) = \gamma(5) = 00101$$

$$L = 4 = 0100_2$$

Concaténer $\gamma(M+1)$ et L, on aura $\delta(N) = 00101 0100$.

b. Deuxième méthode

Cette méthode est purement Delta, le codage se fait selon les étapes suivantes :

1. écrire N en binaire. Le bit le plus significatif (en extrême gauche) sera un 1.
2. Compter les bits, enlever le bit le plus à gauche de N, et ajouter le compte en binaire au début de reste du nombre N après lui avoir enlevé son bit le plus significatif.
3. Soustraire 1 du compte de l'étape 2 et ajouter ce nombre de zéros au début du code.

- **Exemple**

Soit à coder le nombre 20 en utilisant la deuxième méthode de Delta

$$N = 20 = 10100_2 \quad (5 \text{ bits})$$

Enlever le 1 le plus à gauche et ajouter 5 en binaire au début du résultat, ce qui donne 1010100

Trois bits ont été ajoutés, alors on ajoute deux (3-1) zéros pour obtenir le code final Delta : $\delta(20) = 001010100$.

Quelle que soit la méthode de Delta utilisé, les codes obtenus seront les mêmes.

Nombre $N=2^M + L$	Code $\gamma(M)B(L)$	Nombre	Code
$1 = 2^0 + 0$	1	$11 = 2^3 + 3$	00100011
$2 = 2^1 + 0$	0100	$12 = 2^3 + 4$	00100100
$3 = 2^1 + 1$	0101	$13 = 2^3 + 5$	00100101
$4 = 2^2 + 0$	01100	$14 = 2^3 + 6$	00100110
$5 = 2^2 + 1$	01101	$15 = 2^3 + 7$	00100111
$6 = 2^2 + 2$	01110	$16 = 2^4 + 0$	001010000
$7 = 2^2 + 3$	01111	$17 = 2^4 + 1$	001010001
$8 = 2^3 + 0$	00100000	$18 = 2^4 + 2$	001010010
$9 = 2^3 + 1$	00100001	$19 = 2^4 + 3$	001010011
$10 = 2^3 + 2$	00100010	$20 = 2^4 + 4$	001010100

Tableau III.7: Codage Delta des 20 premiers entiers positifs

➤ Décodage

Pour le processus de décompression on suit les étapes ci-après :

1. Lire le code de gauche à droite jusqu'à reconnaître un code Gamma qui représente $M+1$.
2. Lire les M bits suivant qui représente le nombre L .
3. Calculer $N = 2^M + L$.

4.3.1.3. Code Omega $\omega(N)$ [12]

Comme les codes précédents d'Elias (Gamma et Delta), le code Omega est destiné pour coder les nombres entiers positifs sans qu'il y ait besoin au préalable de connaître l'intervalle de valeurs à coder. Mais à la différence des codes précédents d'Elias, ce code s'emploie périodiquement pour coder le préfixe M , c'est pourquoi parfois il est appelé code récursif d'Elias.

➤ Codage

Le processus de codage Omega se fait comme suit :

1. Initialiser le code-ainsi-loin à 0.
2. Si le nombre à coder est 1, arrêter ; Autrement, ajouter au début la représentation binaire de N au code-ainsi-loin (à gauche de 0). Supposer que nous avons ajouté L bits au début.
3. Répéter l'étape 2, avec la représentation binaire de $L-1$ au lieu du N .

• Exemple

Soit à coder le nombre 20 en utilisant le code Omega

- 1) Initialiser le code à 0 i.e. $\omega(N) = 0$.
- 2) $N = 20 = 10100_2$ (5 bits).

Ajouter N au début de $\omega(N)$ i.e. $\omega(N) = 101000$.

- 3) Le nombre N est représenté sur 5 bits en binaire, donc on représente 4 (qui est 5-1) en binaire qui est 100 (sur 3 bits).

Ajouter le nombre binaire 100 au début de $\omega(N)$ i.e. $\omega(N) = 100101000$.

- 4) Le nombre 4 est représenté sur 3 bits en binaire, donc on représente 2 (qui est 3-1) en binaire qui est 10 (sur 2 bits).

Ajouter le nombre binaire 10 au début de $\omega(N)$ i.e. $\omega(N) = 10100101000$.

- 5) On arrête, car le nombre à coder est 1 (2-1). Le résultat est $\omega(N) = 10100101000$.

Nombre	Code	Nombre	Code
1	0	11	11 1011 0
2	10 0	12	11 1100 0
3	11 0	13	11 1101 0
4	10 100 0	14	11 1110 0
5	10 101 0	15	11 1111 0
6	10 110 0	16	10 100 10000 0
7	10 111 0	17	10 100 10001 0
8	11 1000 0	18	10 100 10010 0
9	11 1001 0	19	10 100 10011 0
10	11 1010 0	20	10 100 10100 0

Tableau III.8: Codage Omega des 20 premiers entiers positifs

➤ Décodage

Le décodage se fait en plusieurs étapes non récurrentes où chaque étape lit un groupe de bits du code. Un groupe qui commence par un zéro signale la fin du décodage.

1. Initialiser N à 1
2. Lire le prochain bit. S'il est un 0, arrêter. Autrement lire N bit de plus, affecter le groupe de N + 1 bits à N, et répéter cette étape.

• Exemple

Soit à décoder le code 10 100 10100 0 en utilisant Omega

On a $\omega(N) = 10100101000$

- 1) On initialise N à 1 i.e. $N = 1$.
- 2) Lire le premier bit qui est un 1, alors il lit $N=1$ bit de plus qui est 0 et affecter le groupe 2 ($N+1$) bits à N, ce qui donne $N = 10_2 = 2_{10}$.

- 3) Lire le prochain bit qui est un 1, alors il lit $N=2$ bit de plus qui est 00 et affecter le groupe de 3 bits à N , ce qui donne $N = 100_2 = 4_{10}$.
- 4) Lire le prochain bit qui est un 1, alors il lit $N=4$ bit de plus qui est 0100 et affecter le groupe de 5 bits à N , ce qui donne $N = 10100_2 = 20_{10}$.
- 5) Lire le prochain bit qui est un 0, alors on arrête. Le résultat est $N = 20$.

4.3.2. Codage de Levenstein [14]

Le code de Levenstein est l'un des codes universels utilisé pour le codage des nombres entiers non-négatifs dont la borne supérieure est inconnue. Le mot de code pour un nombre entier n est produit par la fonction L :

$$L : \mathbb{N} \times \mathbb{N} \rightarrow \{0,1\}^*$$

$$L(n) = \begin{cases} 0 & \text{si } n=0 \\ 1L(k)N(n,k) & \text{sinon, où } k = \lfloor \log_2(n) \rfloor \end{cases}$$

$$N : \mathbb{N} \times \mathbb{N} \rightarrow \{0,1\}^*$$

$$N(n,k) = \begin{cases} \varepsilon & \text{si } k=0 \\ N(\lfloor n/2 \rfloor, k-1)b & \text{sinon, où } b = n \bmod 2 \end{cases}$$

Les codes des 10 premiers nombres entiers sont représenté dans le tableau suivant :

Nombre entier	Code de Levenstein
0	0
1	10
2	110 0
3	110 1
4	1110 0 00
5	1110 0 01
6	1110 0 10
7	1110 0 11
8	1110 1 000
9	1110 1 001

Tableau III.9: Codage Levenstein des 10 premiers entiers positifs

Pour le décodage, il suffit de :

1. Lire c 1 jusqu'à ce qu'un 0 soit lu (tel que c est le nombre de bits 1 lus).
2. Si $c=0$, alors le nombre est $n=0$.
3. Sinon, initialiser la variable $n=1$ et répéter $c-1$ fois :
 - 3.1 lire n nouveaux bits, ajouter un 1 à gauche des bits lus et assigner le résultat à n .

5. Conclusion

Dans ce chapitre nous avons présentés la méthode de compression VLC, quelques définitions et les techniques de codage à longueur variable.

Les techniques de codage à longueur variable permettent souvent de compresser d'une manière très efficace, surtout dans les cas où une grande partie des symboles d'un fichier sans répétés.

Chapitre IV

Analyse &

Conception

1. Introduction

Dans ce chapitre on va d'abord présenter le code VLC proposé, ensuite expliquer d'une façon détaillée l'analyse et la conception du système de compression / décompression en présentant ainsi le processus de codage et de décodage des données. Ce système doit fournir en plus des fonctionnalités de compression/décompression d'autres fonctions telles que le calcul de temps, le taux, la taille des fichiers.

2. Le code VLC proposé

Le code proposé est composé de deux parties, une partie fixe et une partie variable.

2.1. Codage

Le codage fixe est facile à générer. Pour un code fixe sur N bits on aura 2^N Symboles $S = \{S_0, S_1, \dots, S_{2^n-1}\}$.

Le codage VLC correspondant à cet ensemble de symboles est :

$$C(S_i) = (prefix)_i . (suffix)_i / i \in [0, 2^n - 1]$$

$C(S_i)$: Codage du symbole S_i .

Les deux parties constituant le code proposé sont :

2.1.1. Prefix

Cette partie représente deux informations qui sont :

1. La première information est la position (P) du bit le plus significatif dans le code fixe de S_i . Sa valeur va de 0 à N-1

$$Val(prefix) = P \in [0, N - 1]$$

Cas particulier : Pour $S_i = 0 \rightarrow Prefix = 0$

- **Exemple**

$$S_8 = (8)_{10} = (\underbrace{00 \dots \dots 1000}_{N \text{ bits}})$$

La valeur du champ *prefix* est la position du bit le plus significatif de la droite à gauche. $Val(prefix) = P = 3$.

2. La deuxième information est la longueur de la partie *suffix*

La longueur de la partie prefix : C'est le nombre de bits de la partie *prefix*.

Pour représenter un nombre X , le nombre de bits n nécessaire se calcule comme suit :

$$n = 1 + \log_2 X \approx \log_2(X + 1) \dots (1)$$

Pour le calcul de la longueur de la partie *prefix*, il est indispensable de trouver la taille de la plus grande valeur du *prefix* appartenant à l'intervalle $[0, N - 1]$ qui est $X = N - 1$.

En appliquant la règle (1), on aura :

$$L(\text{prefix}) = L(N - 1) = \log_2((N - 1) + 1) = \log_2(N) \text{ bits.}$$

Tel que $L(\text{prefix})$ est la longueur du *prefix* en nombre de bits.

$$L(\text{prefix}) = \log_2(8) = \frac{\log(8)}{\log(2)} = 3 \text{ bits}$$

2.1.2. Suffix

C'est la partie restante du code fixe S_i qui suit le bit significatif.

1. $\text{Val}(\text{suffix}_i) = S_i - 2^{\text{prefix}_i} \rightarrow S_i = 2^{\text{prefix}_i} + \text{suffix}_i$
2. $L(\text{suffix}_i) = \begin{cases} 1 & \text{si } \text{prefix}_i = 0 \\ \text{prefix} & \text{sinon} \end{cases}$

- **Exemple 2**

$N=8$ bits, le nombre de symboles est $2^N = 2^8 = 256$.

En reprenant le même code S_6 de l'Exemple 1 et on retrouve la valeur du *suffix*.

$$S_8 = \left(\underbrace{00 \dots 1}_{N \text{ bits}} \underbrace{000}_{\text{suffix}} \right)_2, \text{ de là on a :}$$

- $\text{val}(\text{suffix}) = 0$
- $L(\text{suffix}) = \text{prefix} = 3 \text{ bits}$

Des exemples 1 et 2, on peut conclure le code VLC correspondant au symbole

$$S_8 = (00 \dots 1000)_2 \rightarrow C(S_8) = \text{prefix}_8. \text{suffix}_8 = \underbrace{011}_{\text{prefix}} \underbrace{000}_{\text{suffix}} = 011000$$

- **Exemple 3**

$N=4$ bits, $S_{i=1,15}$ l'ensemble des symboles.

Compression de la chaîne $CH=S_2S_2S_3S_2S_4S_3$.

La représentation binaire de CH est 0010 0010 0011 0010 0100 0011 et la taille de CH est : $L(CH)=4 \times 6=24$ bits. La compression consiste à remplacer chaque code de taille fixe 4 bits par son code VLC correspondant suivant les étapes ci-après :

- Calculer les probabilités des symboles et les triées selon l'ordre décroissant :

Indice	Symboles (S_i)	Probabilité $P(S_i)$
0	S_2	0.5
1	S_3	0.33
2	S_4	0.16

- Construire le code VLC

Au lieu de coder les symboles eux-mêmes, on procède au codage des indices des symboles.

$$C(S_i) = Prefix_i \cdot Suffix_i$$

$$N = 4 \text{ bits} \rightarrow L(prefix_i) = \log_2 4 = 2 \text{ bits}$$

Indice	Indice binaire	Position du bit poids fort	Code C (s_i)	
			Prefix	Suffix
0	0000	0	00	0
1	0001	0	00	1
2	0010	1	01	0

- Remplacer chaque code fixe par son code préfixé

$$C(CH) = CH_c = 000\ 000\ 001\ 000\ 010\ 001.$$

CH_c : chaîne compressée. $L(CH_c) = 3 \times 6 = 18$ bits.

La moyenne de bits par symbole :

$$\sum_{i=1}^N P(s_i) L(c_i) = 0.5 \times 3 + 0.33 \times 3 + 0.16 \times 3 = 2.99$$

$$\sum_{i=1}^N P(s_i) L(c_i) \approx 3 \text{ bits/symbole.}$$

2.2. Décodage

Concernant le processus de décodage, la taille du code initial (N) et la table des probabilités des différents symboles doivent être connues au préalable. Puis procéder à la décompression suivant les étapes ci-après:

- Lire $\log_2 N$ premier bits (*prefix*)
- Si *prefix* = 0 , lire 1 bit
Sinon lire *prefix* bits (*suffix*)
- Calculer *indice* = $2^{\text{prefix}} + \text{Suffix}$.
- S_i = Table [*indice*] et le représenter sur N bits

2.3. Longueur du code

La longueur de chaque code VLC assigné aux codes fixe sur N bits est calculée par la relation suivante :

$$\begin{aligned} L(C_i) &= L(\text{prefix}_i . \text{suffix}_i) = L(\text{prefix}_i) + L(\text{suffix}_i) \\ &= \log_2(N) + \text{prefix}_i \end{aligned}$$

Nous avons : $L(\text{suffix}_i) = \begin{cases} 1 & \text{si } \text{prefix}_i = 0 \\ \text{prefix}_i & \text{sinon} \end{cases}$

Donc : $L(C_i) = \begin{cases} \log_2(N) + 1, & i = 0 \\ \log_2(N) + \text{prefix}_i, & \forall i \in [1, 2^N - 1] \end{cases} \dots(2)$

$L(C_i) = Cte + \text{prefix}_i$, avec $Cte = \log_2(N)$

Et sachant que: $\text{prefix}_i \in [0, N - 1] \rightarrow$ Le nombre de taille du codes est

N différentes tailles, Alors (2) devient :

$$L(C_i) = \begin{cases} \log_2(N) + 1, & i = 0 \\ \log_2(N) + \text{prefix}_i, & \text{prefix}_i \in [1, N - 1] \end{cases}$$

2.4. Liste des codes VLC

La liste des codes VLC correspondants aux codes fixes représentés sur N bits est présentée dans le tableau suivant :

Indice	Symbole S_i	bit fort	Code C_i		$L(C_i)$	Nombre de code $[2^{L(suffix)}]$
			<i>Prefix</i>	<i>Suffix</i>		
0 :1	$\overbrace{000\dots\dots X}^{N \text{ bits}}$	0	$\overbrace{000\dots\dots 0}^{Log2(n)}$	X	$\log_2(n)+1$	$2^1=2$
2 :3	000.....1X	1	000.....1	X	$\log_2(n)+1$	$2^1=2$
4 :7	000.....1XX	2	000.....10	XX	$\log_2(n)+2$	$2^2=4$
$1 : 2^{n-1}-1$	01X XXX	N-2	111.....110	X.....XXX	$\log_2(n)+n-2$	2^{n-2}
$2^{n-1} : 2^n-1$	1XX...XXX	N-1	111.....111	XX...XXX	$\log_2(n)+n-1$	2^{n-1}

Tableau IV.1 : Tableau des codes VLC correspondants aux codes fixes sur N bits

Remarque : pour un ensemble de symboles su N bits, on a les résultats suivants :

- 2^N différents codes VLC.
- $L(C)_i = \log_2(N) + prefix_i, prefix_i \in [1, N - 1]$, il ya N tailles différentes.
- Le nombre de code pour chaque longueur est 2^{prefix_i}

• **Exemple 4**

Supposons $n=4$ bits, il y a $2^4=16$ symboles. Construisons le code VLC correspondant.

$$C(S_i) = prefix_i . suffix_i, \forall i \in [0, 2^4 - 1]$$

Calculer le nombre de bits de la partie *prefix* :

$$L(prefix_i) = \log_2(n) = \log_2(4) = \frac{\log(4)}{\log 2} = 2 \text{ bits.}$$

Les codes VLC correspondants aux codes fixes représentés sur 4 bits est présentée dans le tableau suivant :

Indice	Symbole S_i	bit poids fort	Code C_i		$L(C_i)$ $\log_2(8)+p$	Nombre de code $[2^{L(suffix)}]$
			Prefix	Suffix		
0-1	000X	0	000	X	4	2
2-3	001X	1	001	X	4	$2^1=2$
4-7	01XX	2	010	XX	5	$2^2=4$
8-15	1XXX	3	011	XXX	6	$2^3=8$

Tableau IV.2 : Tableau des codes VLC correspondants aux codes fixes sur 4 bits

2.5. Démonstration de l'unicité du code VLC

L'unicité de décodage signifie qu'aucun code n'est préfix d'un autre code.

Supposons qu'ils existent deux codes C_1 et C_2 du code VLC proposé qui ne respectent pas cette propriété.

Sachant que :

$$C_1 = P_1.S_1 \dots \dots (1)$$

$$C_2 = P_2.S_2 \dots \dots (2)$$

Avec P_i : Prefix_i, S_i : Suffix_i.

Alors C_2 est un préfix de C_1 implique $C_2=C_1.X$ (C_2 est la concaténation du C_1 avec un $X \neq \emptyset$).

On remplace C_1 par sa valeur $\rightarrow C_2=P_1 . S_1 .X \dots \dots (3)$

De (2) et (3) $\rightarrow C_2 = P_2.S_2 = P_1.S_1.X \dots (4)$

En appliquant l'opérateur de longueur :

$$L(P_2.S_2) = L(P_1.S_1.X) \rightarrow L(P_2) + L(S_2) = L(P_1) + L(S_1.X)$$

On a aussi $L(P_1)=L(P_2)=P=\log_2(n)$ c'est une constante. Donc :

$$L(S_2) = L(S_1.X) \dots (5)$$

Nous pouvons déduire à partir :

De (1) que $L(S_1)=P_1$ (longueur de la partie suffixe du code C_1)

De (2) que $L(S_2)=P_2$ (longueur de la partie suffixe du Code C_2)

Deux cas se présentent :

1^{er} cas : $P_1 \neq P_2 \rightarrow C_1$ n'est pas un préfix du code C_2

2^{eme} cas : $P_1 = P_2$ l'équation (5) devient

$$L(S_2) = L(S_1 \cdot X) \leftrightarrow L(S_2) = L(S_1) + L(X)$$

$$\leftrightarrow P_2 = P_1 + L(X),$$

$$P_1 = P_2 \rightarrow L(X) = 0 \rightarrow C_2 = C_1$$

Cela nous mène à déduire que le code VLC proposé est un code préfixé et assure l'unicité de décodage.

2.6. Vérification du théorème de Kraft-McMillan

Si on applique l'inégalité de Kraft sur notre code on aura :

$$\begin{aligned} \sum_{i=0}^{2^N-1} 2^{-L(C_i)} &= \frac{1}{2^{L(C_0)}} + \frac{1}{2^{L(C_1)}} + \dots + \frac{1}{2^{L(C_{2^N-1})}} \\ &= \frac{1}{2^{L(C_0)}} + \frac{1}{2^{L(C_1)}} + \dots + \frac{1}{2^{L(C_{2^N-1})}} \dots (1) \end{aligned}$$

Dans le code nous avons :

- 2^N codes VLC
- N longueurs de codes
- Pour chaque Longueur $L(C)_j, j \in [1, N-1]$, il existe 2^{prefix_j} codes, ($j=0, L(C_0)=2$).

$$(1) = \sum_{j=0}^{N-1} 2^{prefix_j} \times 2^{-L(C)_j} = \sum_{j=0}^{N-1} \frac{2^{prefix_j}}{2^{L(C)_j}}$$

On a : $L(C)_j = L(prefix)_j + L(suffix)_j = \log_2(N) + prefix_j$

$$\sum_{j=0}^{N-1} \frac{2^{prefix_j}}{2^{L(C)_j}} = \sum_{j=0}^{N-1} \frac{2^{prefix_j}}{2^{(\log_2(N) + prefix_j)}}$$

$$\begin{aligned}
&= \sum_{j=0}^{N-1} \frac{2^{\text{prefix}_j}}{2^{\log_2(N)} \times 2^{\text{prefix}_j}} \\
&= \sum_{j=0}^{N-1} \frac{1}{2^{\log_2(N)}} = \frac{N}{2^{\log_2(N)}} = \frac{N}{N} = 1
\end{aligned}$$

D'où : $\sum_{i=0}^{2^N-1} 2^{-L(C_i)} \leq 1$ (L'inégalité de Kraft est vérifiée)

On voit bien que le code VLC proposé vérifie l'inégalité de Kraft quel que soit la valeur de N.

Alors on déduit que ce code est préfixé.

2.7. Avantage du code VLC proposé

Le code proposé est caractérisé par :

- **La simplicité** : la génération du code est très simple et ne nécessite pas de modèles mathématiques compliqués, est aussi instantanément déduit du code fixe correspondant. On peut même l'implémenter sur des supports hardware puisque le codage et le décodage s'effectuent par des instructions de décalages.
- **La flexibilité** : le code est paramétrable. Plusieurs variétés du code peuvent être obtenues selon les paramètres : N, *prefix*, *suffix*.

3. Cadre de travail

Notre travail consiste à combiner les deux méthodes existantes LZW et VLC pour assurer une compression temps réel des données.

Afin d'atteindre l'objectif de notre travail, nous avons considéré la modularité comme un aspect fondamental à la conception de l'application. Il permet d'une part de répondre aux besoins fixés, et d'autre part, de faciliter le développement et la maintenance de ce système ou de cette application.

4. Principe générale de la méthode LZW_VLC proposé

Le principe de la compression et décompression de la méthode (LZW_VLC) est illustré dans la figure IV.1:

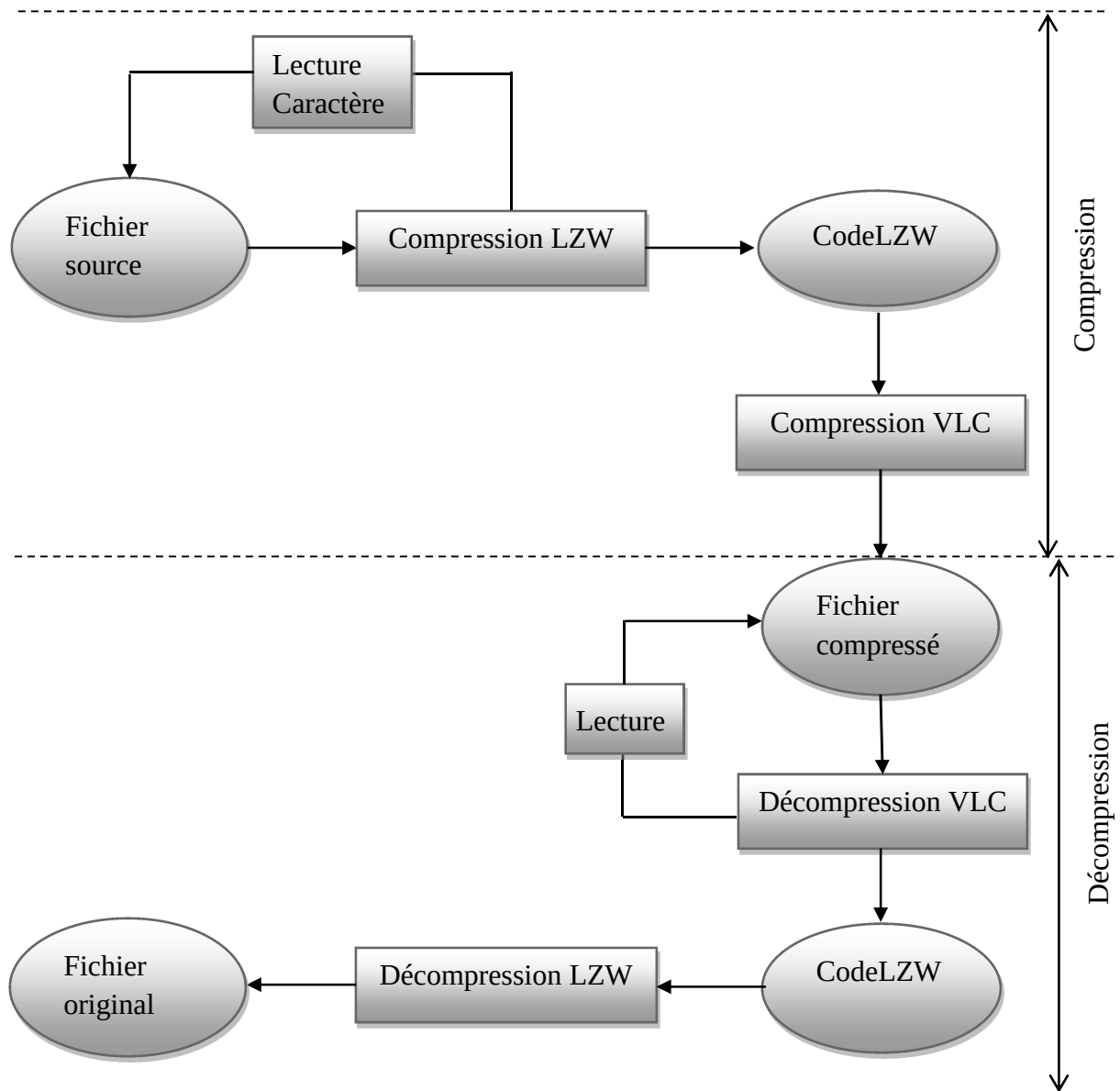


Figure IV.1 : Schéma générale de compression/décompression LZW_VLC

5. Modules et Algorithmes principaux de l'application

Avant de donner les différents modules qui composent ce système, il est nécessaire de donner les grandes lignes aux quelles ils feront référence, cela afin d'obéir à l'objectif de l'application à créer.

Pour ce faire notre application doit alors offrir la possibilité de gérer les différentes tâches et demandes de l'utilisateur, qu'elle doit accomplir à savoir la compression, la

décompression, le calcul et l'affichage des résultats et performances de compression / décompression.

Donc nous pouvons décomposer le noyau de l'application en deux modules principaux, (voir figure IV.2) :

- Module de compression.
- Module de décompression.

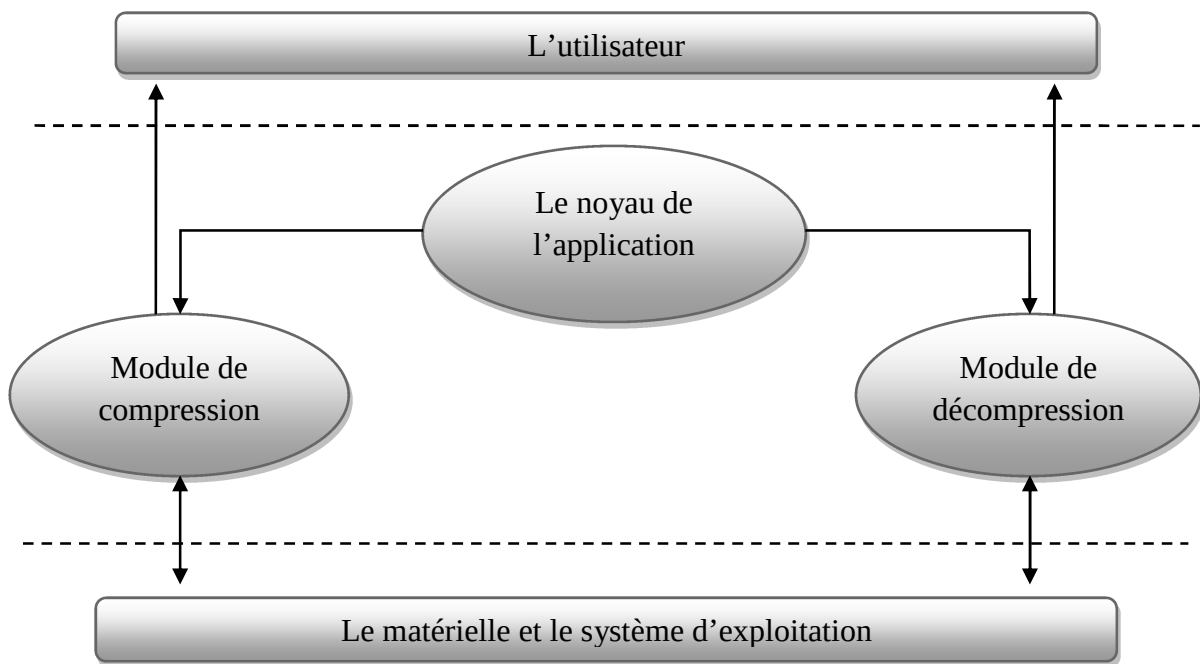


Figure IV.2 : Décomposition des modules principaux de l'application

5.1. Algorithmes du module de compression

Après avoir spécifié le fichier à compresser, le rôle principale de la fonction de compression est d'écrire l'entête dans le fichier de sortie, construire une table des symboles, construire un dictionnaire où est placée chaque nouvelle séquence avec son adresse. Et au fur et à mesure, pour chaque adresse (symbole) vérifier si elle existe déjà dans la table des symboles, l'ajouter à la table sinon. Réordonner cette table par ordre décroissant des fréquences apparition des symboles. Enfin coder les indices des symboles se trouvant dans cette table. Refaire ces étapes jusqu'à la fin du fichier source.

5.1.1. Algorithmes générale de la compression

Début

Ouvrir le fichier source
Créer un nouveau fichier
Initialiser le dictionnaire
Initialiser la table d'index
Ecrire l'entête du fichier de sortie
Compresser ()
Fermer le fichier source
Fermer le fichier de sortie

Fin.

Cet algorithme est illustré dans la figure IV.3.

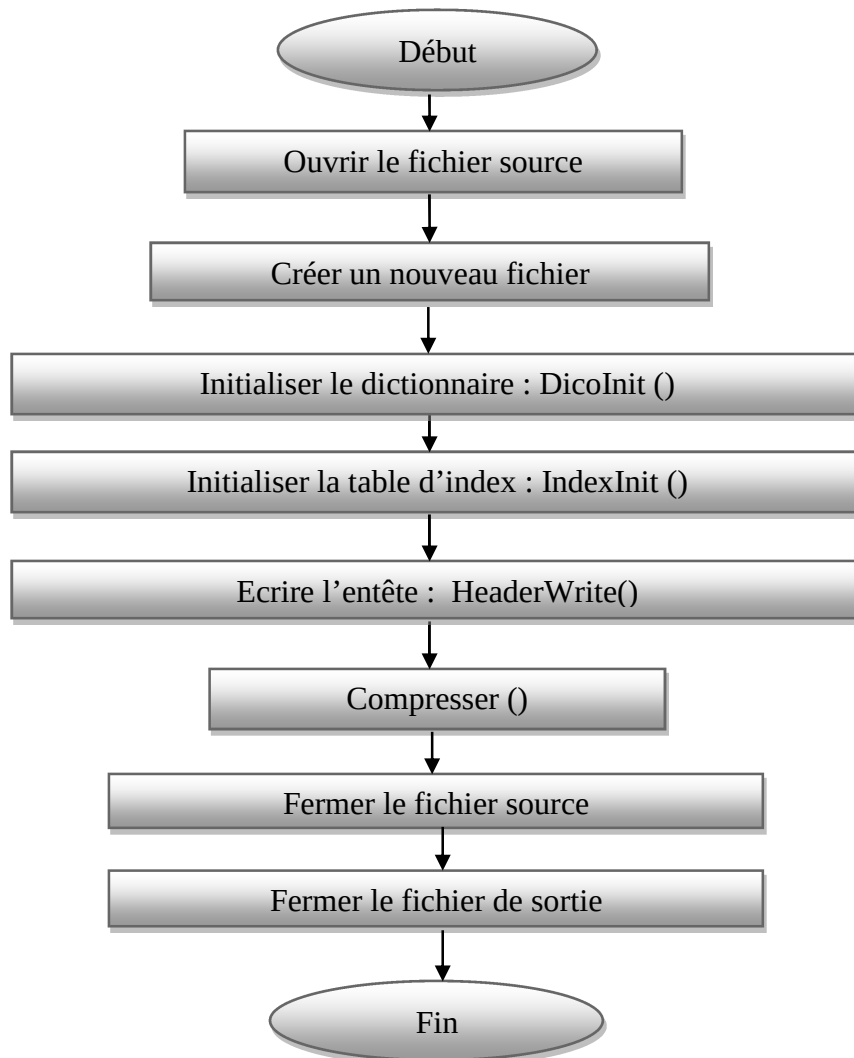


Figure IV.3 : Organigramme illustrant les étapes de compression.

5.1.2. Algorithme de la fonction compresser ()

Début

Récupérer le premier caractère S

Tant que ce n'est pas la fin du fichier faire

 Récupérer un caractère T

Si (U :=ST) ∈ Dictionnaire

 Récupérer son code du dictionnaire

 Code_LZW := code de U

Sinon

 Ajouter U au dictionnaire

 Envoyer Code_LZW

 CoderEnVLC (Code_LZW)

 S:=T

Finsi

Fin tant que

Fin.

Cet algorithme est illustré dans la figure IV.4.

5.1.3. Algorithme de la fonction CoderEnVLC (Code_LZW)

Début

Si code_lzw appartient à la table

 Récupérer sa position

 Incrémenter sa fréquence

 Réordonner la table

 Ecrire '1' dans le fichier de sortie

 Récupérer le Code_VLC du Code_LZW

 Ecrire le Code_VLC

Sinon

 Ajouter le Code_LZW à la table

 Mettre la fréquence du Code_LZW à 1

 Incrémenter la dernière position dans la table

 Ecrire '0' dans le fichier de sortie

 Ecrire le Code_LZW sur NBBIT dans le fichier de sortie //NBBIT==12 bit

finsi

fin.

Cet algorithme est illustré dans la figure IV.5.

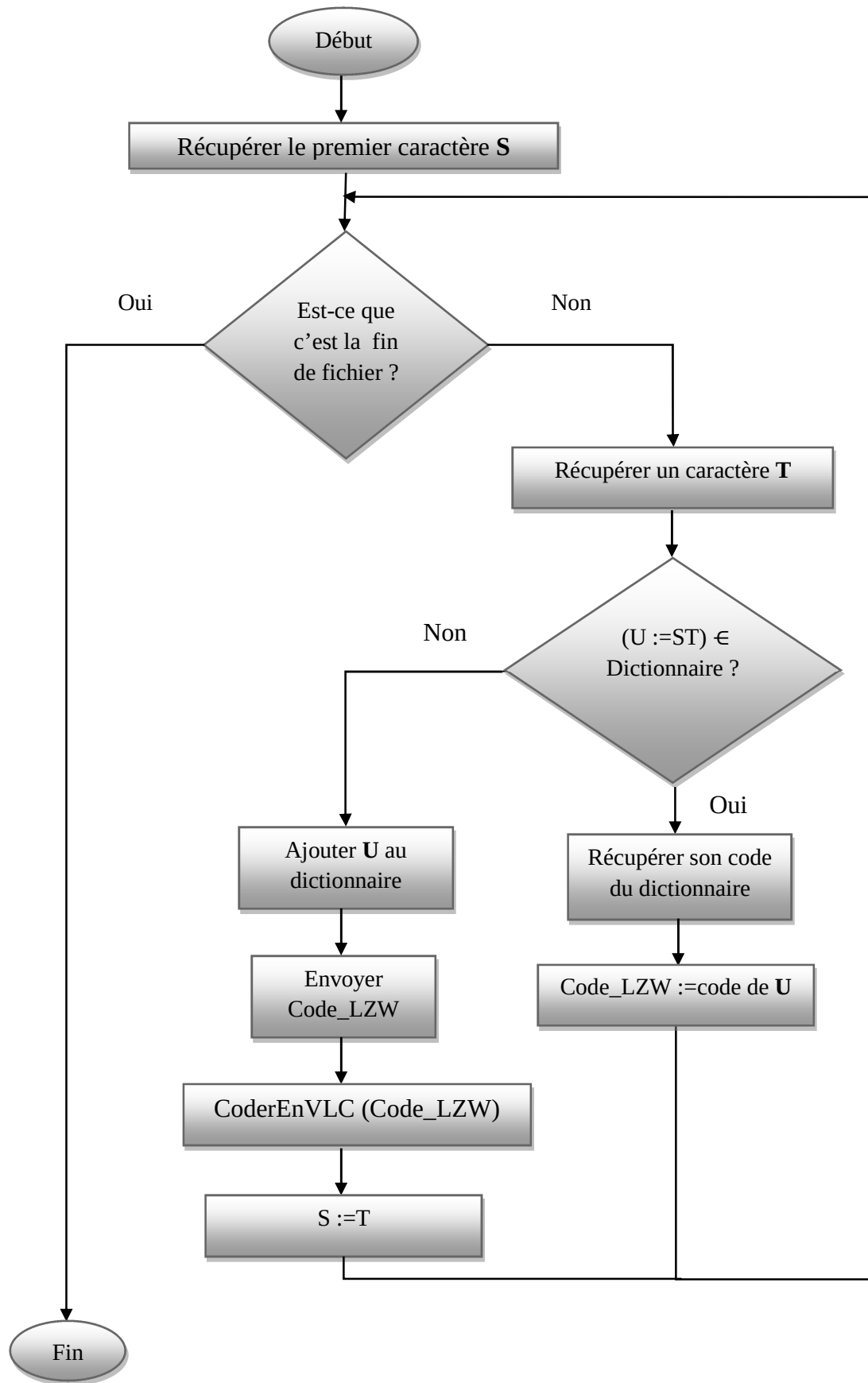


Figure IV.4 : Organigramme illustrant les étapes de la fonction compresser ().

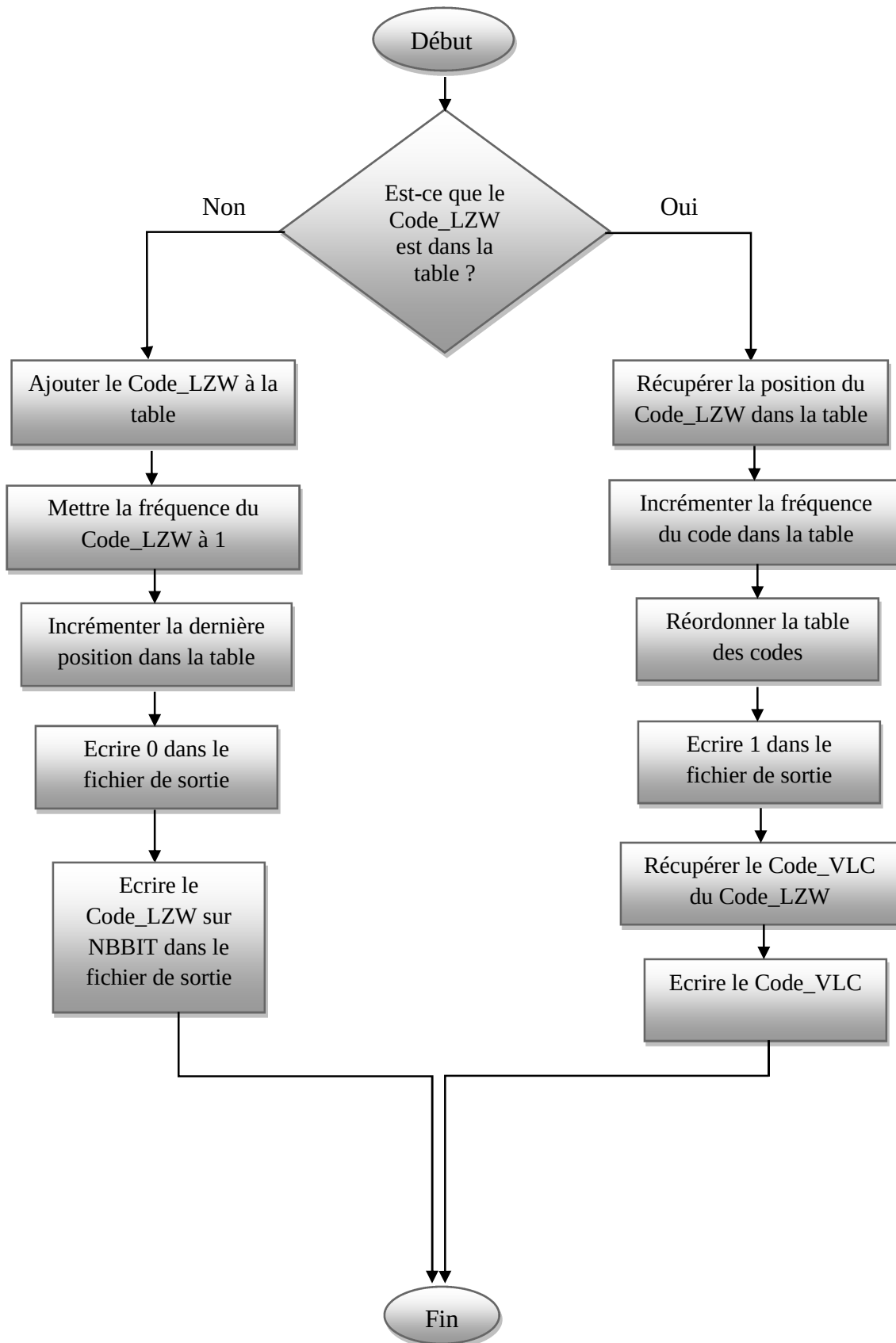


Figure IV.5 : Organigramme illustrant les étapes de la fonction CoderEnVLC (Code_LZW).

- **Exemple 4**

Soit à compresser la chaîne « abababbaba »

Les instructions essentielles de cet algorithme se dérouleront comme suit :

début

- 1) String_code := code(a) // code (a) = 97
- 2) **Tant que** ce n'est pas la fin de la chaîne **faire**
 1. Character := code (b) // code(b) = 98
 - a) U := 'ab'
 - b) Ajouter 'ab' au dictionnaire //adresse(ab) = 256
 - c) Envoyer 97
 - a. Ajouter 97 à la table
 - b. Fréquence de 97 à 1
 - c. Ecrire 0
 - d. Ecrire 000001100001
 - d) String_code :=code(b)
 2. Character := code (a) // code(a) = 97
 - a) U := 'ba'
 - b) Ajouter 'ba' au dictionnaire //adresse(ba) = 257
 - c) Envoyer 98
 - a. Ajouter 98 à la table
 - b. Fréquence de 98 à 1
 - c. Ecrire 0
 - d. Ecrire 000001100010
 - d) String_code :=code(a)
 3. Character := code (b)
 - a) U := 'ab'
 - b) String_code :=code(ab)
 4. Character := code (a)
 - a) U := 'aba'
 - b) Ajouter 'aba' au dictionnaire //adresse(aba) = 258
 - c) Envoyer 256
 - a. Ajouter 256 à la table
 - b. Fréquence de 256 à 1
 - c. Ecrire 0
 - d. Ecrire 000100000000
 - d) Sting_code :=code(a)
 5. Character := code (b)
 - a) U := 'ab'
 - b) Sting_code :=code(ab)
 6. Character := code (b)
 - c) U := 'abb'
 - d) Ajouter 'abb' au dictionnaire //adresse(abb) = 259

- e) Envoyer 256
 - a. Fréquence de 256 à 2
 - b. Ecrire 1
 - c. Ecrire 00
 - f) Sting_code :=code(b)
 - 7. Character := code (a)
 - a) U := 'ba'
 - b) String_code :=code(ba)
 - 8. Character := code (b)
 - a) U := 'bab'
 - b) Ajouter 'bab' au dictionnaire //adresse(bab) = 260
 - c) Envoyer 257
 - a. Ajouter 257 à la table
 - b. Fréquence de 257 à 1
 - c. Ecrire 0
 - d. Ecrire 0001000000001
 - d) Sting_code :=code(b)
 - 9. Character := code (a)
 - a) U := 'ba'
 - b) String_code :=code(ba)
 - a. Fréquence de 257 à 2
 - b. Ecrire 1
 - c. Ecrire 01
- 3) **Fin tant que.**

Fin.

Le code de sortie est :

0000001100001000000110001000001000000001000000100000001101

5.2. Algorithmes du module de décompression

Après avoir spécifié le fichier à décompresser, le rôle principale de la fonction de décompression est de : lire l'entête et récupérer les informations qu'il contient, construire une table des symboles, construire un dictionnaire où est placée chaque nouvelle séquence avec son adresse. Puis commencer le processus du décodage en décodant les code avec VLC en reconstruisant la table des symboles, et au fur et à mesure, pour chaque symbole vérifier si il existe déjà dans le dictionnaire, et reconstruire le dictionnaire. Refaire ces étapes jusqu'à la fin du fichier source.

5.2.1. Algorithme générale de la décompression

Début

Ouvrir le fichier source
Créer un nouveau fichier
Initialiser le dictionnaire
Initialiser la table d'index
Lire l'entête du fichier source
Décompresser ()
Fermer le fichier source
Fermer le fichier de sortie

Fin.

Cet algorithme est illustré dans la figure IV.6.

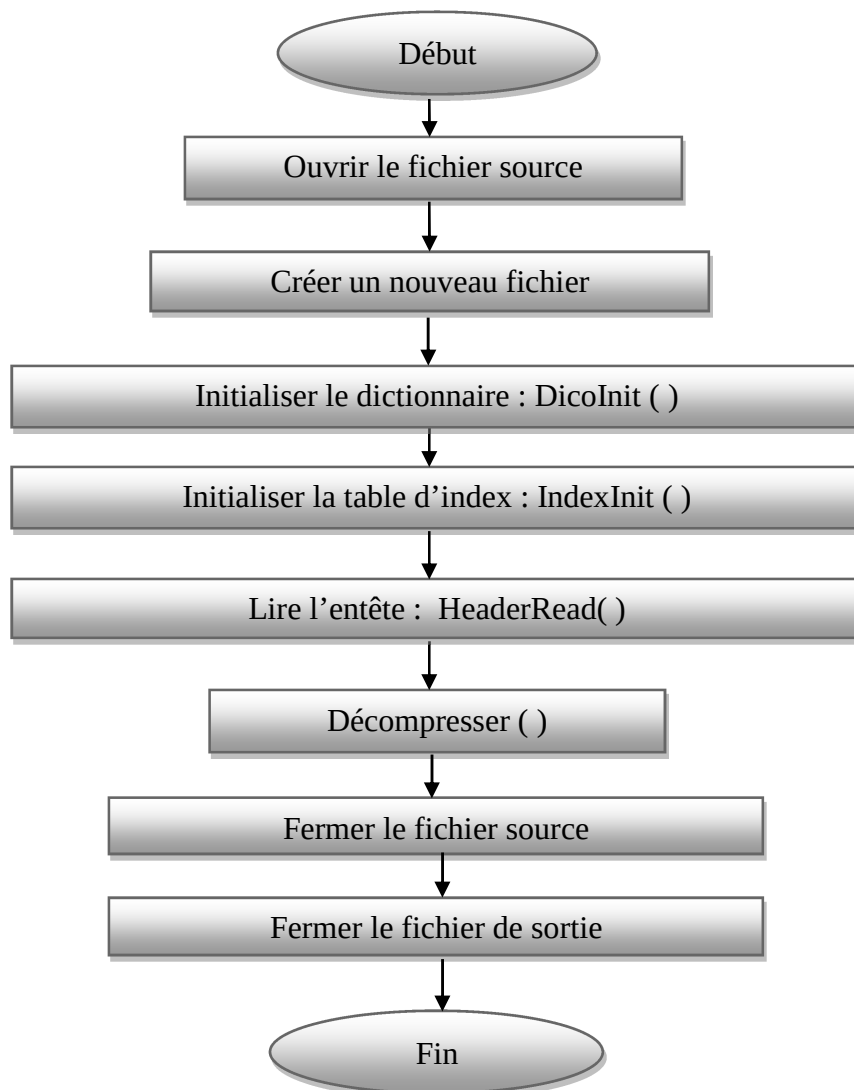


Figure IV.6 : Organigramme illustrant les étapes de décompression.

5.2.2. Algorithme de la fonction Décompresser ()**Début**

Lire un bit

Si (bit == 0)

symb := 0

Tant que (i < NBBIT) **faire**

Lire un bit

symb := (symb<<0) +bit

fin tant que

ajouter symb à la table

mettre la fréquence de symb à 1

adr_préc := symb

ecrire (adr_préc)

fin si

NbCar --

Tant que (NbCar) **faire**

Lire un bit

Si (bit == 0)

DécodeNewCode()

Sinon

DécodeCode()

fin si

NbCar --

fin tant que**fin.**

Cet algorithme est illustré dans la figure IV.7.

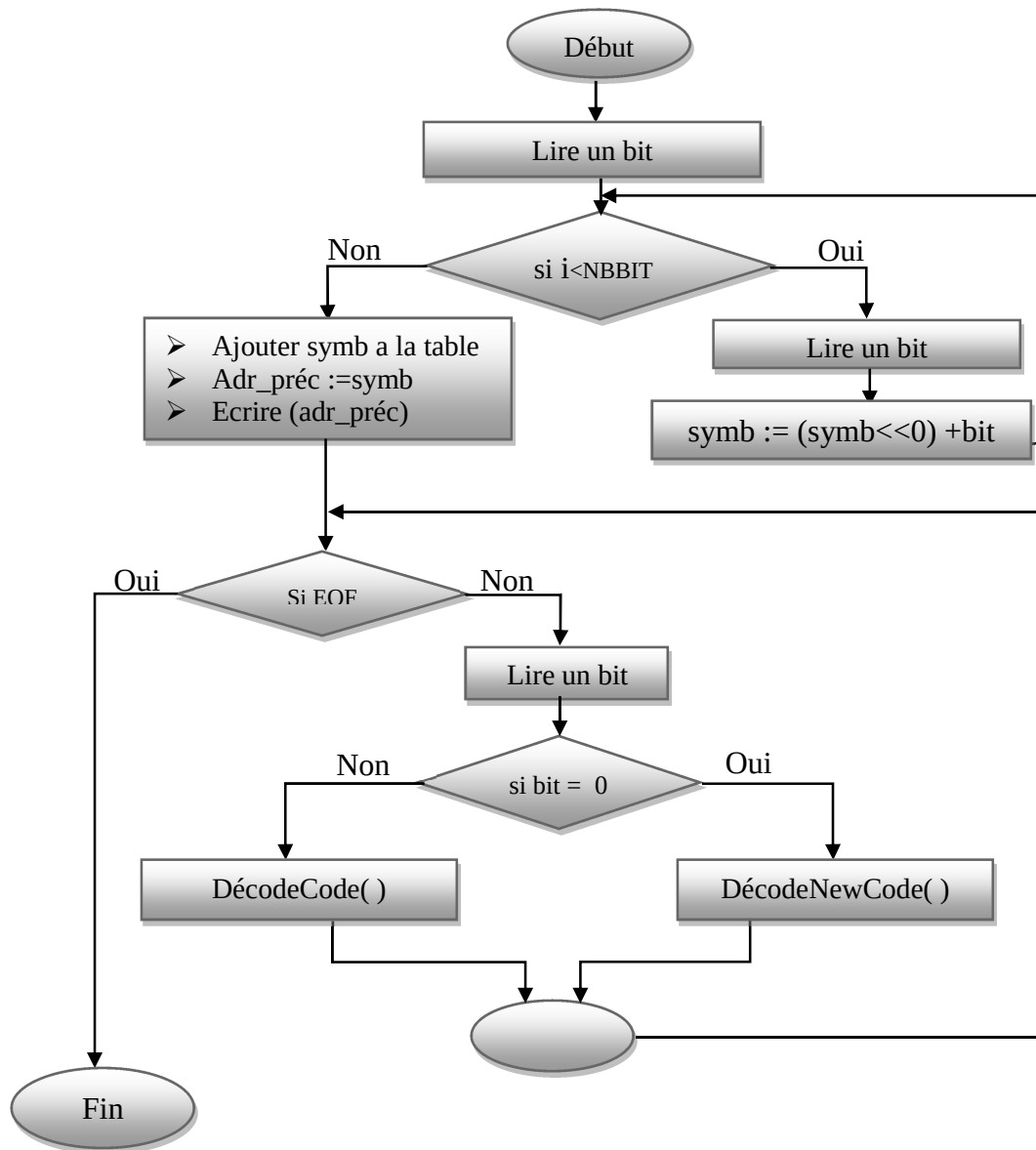


Figure IV.7 : Organigramme illustrant les étapes de la fonction Décompresser ().

5.2.3. Algorithme de la sous fonction DécodeNewCode()

Début

Tant que (i < NBBIT) **faire**

 Lire un bit

 symb := (symb << 0) + bit

Fin tant que

Fin.

Cet algorithme est illustré dans la figure IV.8.

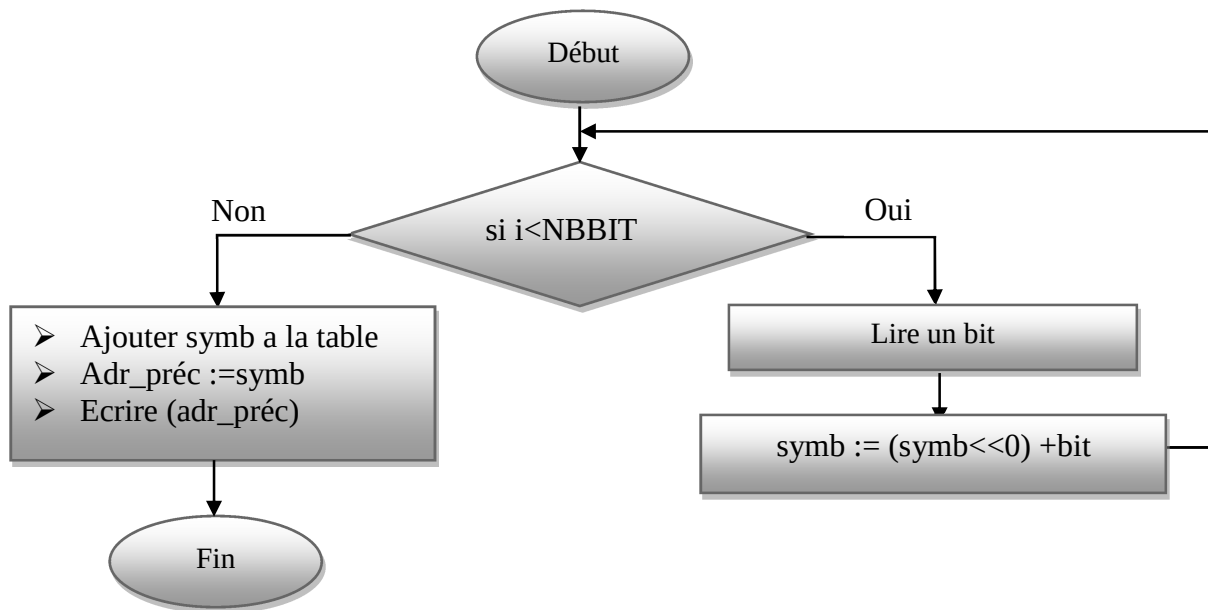


Figure IV.8 : Organigramme illustrant les étapes de la sous fonction DécodeNewCode ().

5.2.4. Algorithme de la sous fonction DécodeCode()

Début

Tant que (i < taille) **faire**

Lire un bit

preffix := (preffix << 1) + bit

fin tant que

if (preffix == 0)

lire un bit

Indexcode := bit

Sinon

Tant que (i < preffix) **faire**

Lire un bit

suffix := (suffix << 1) + bit

fin tant que

Indexcode = (unsigned int)pow(2, preffix) + suffix

Fin si

symb = Table[Indexcode].car

adr_suiv := symb

DécodeEnLZW(adr_prec , adr_suiv)

Réordonner la table

Fin.

Cet algorithme est illustré dans la figure IV.9.

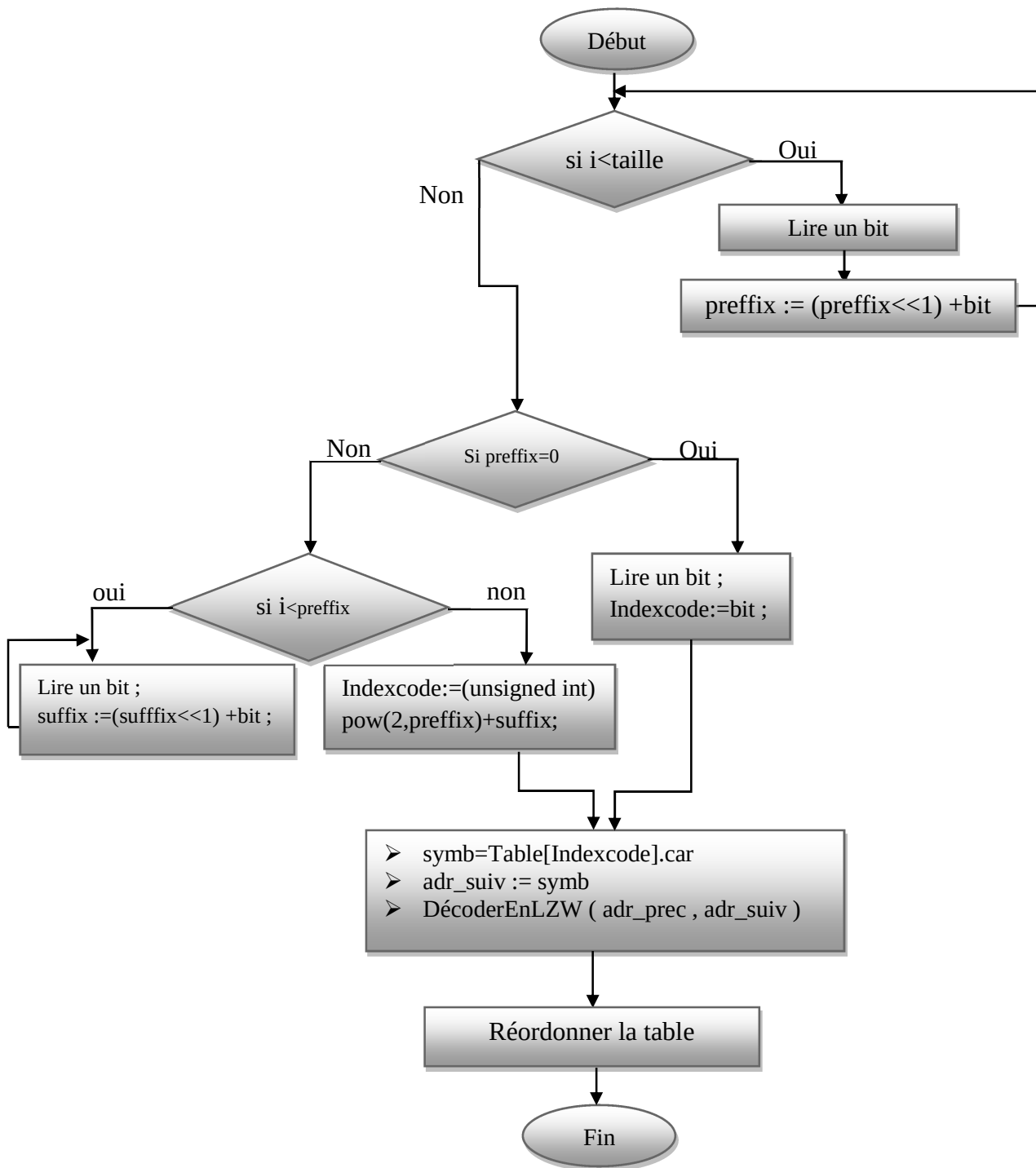


Figure IV.9 : Organigramme illustrant les étapes de la sous fonction DécodeurCode ().

5.2.5. Algorithme de la fonction DécodeEnLZW (adr_préc , adr_suiv)

Début

```

si (adr_suiv ∈ Dictionnaire)
    séq := séquence(adr_suiv)
sinon
    séq := concaténation (adr_préc , t)
fin si
    écrire séquence (séq)
    t := séq[0]
    ajouter (Séquence(adr_préc)+t) au Dico
    adr_préc := adr_suiv
  
```

Fin.

Cet algorithme est illustré dans la figure IV.10.

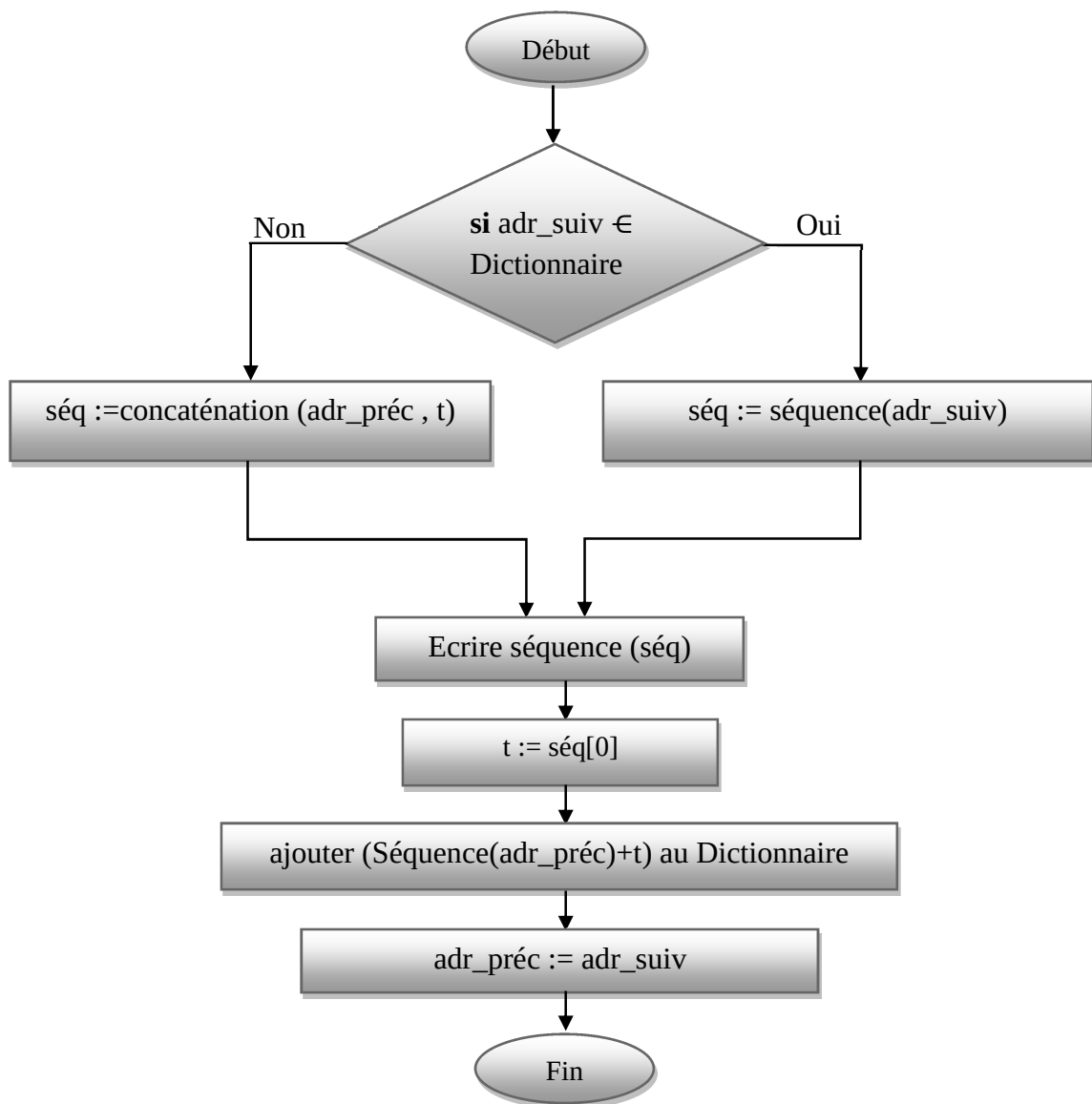


Figure IV.10 : Organigramme illustrant les étapes de la fonction DécodeEnLZW (adr_préc,adr_suiv).

• **Exemple 5**

Soit à décoder le code suivant :

0000001100001000000110001000001000000001000000100000001101

Les instructions essentielles de cet algorithme se dérouleront comme suit :

Début

1) Lire 0

- a) Lire 16 bit
- b) Symb := 97
- c) Ajouter 97 à la table
- d) Mettre la fréquence de 97 à 1
- e) Adr_préc := 97
- f) Ecrire 'a'

2) Tant que ce n'est pas la fin du code faire

I. Lire 0

- a) Lire 12 bit
- b) Symb := 98
- c) Ajouter 98 à la table
- d) Mettre la fréquence de 98 à 1
- e) Adr_suiv := 98
 - i. Séq := 'b'
 - ii. Ecrire 'b'
 - iii. Ajouter 'ab' au dictionnaire //adresse(ab) = 256
 - iv. Adr_préc := 98

II. Lire 0

- a) Lire 12 bit
- b) Symb := 256
- c) Ajouter 256 à la table
- d) Mettre la fréquence de 256 à 1
- e) Adr_suiv := 256
 - i. Séq := 'ab'
 - ii. Ecrire 'ab'
 - iii. Ajouter 'ba' au dictionnaire //adresse(ba) = 257
 - iv. Adr_préc := 256

III. Lire 1

- a) Lire 1 bit
- b) Prefix = 0
- c) Lire 1 bit
- d) Suffix = 0
- e) Symb := 256
- f) Adr_suiv := 256
 - i. Séq := 'ab'
 - ii. Ecrire 'ab'
 - iii. Ajouter 'aba' au dictionnaire //adresse(aba) = 258

- iv. Adr_préc := 256
- IV. Lire 0
 - a) Lire 12 bit
 - b) Symb :=257
 - c) Ajouter 257 à la table
 - d) Mettre la fréquence de 257 à 1
 - e) Adr_suiv :=257
 - i. Séq := 'ba'
 - ii. Ecrire 'ba'
 - iii. Ajouter 'abb' au dictionnaire // adresse(abb) = 259
 - iv. Adr_préc := 257
- V. Lire 1
 - a) Lire 1 bit
 - b) Prefix =1
 - c) Lire 1 bit
 - d) Suffix = 1
 - e) Symb := 257
 - f) Adr_suiv :=257
 - i. Séq := 'ba'
 - ii. Ecrire 'ba'
 - iii. Ajouter 'bab' au dictionnaire //adresse(bab) = 260
 - iv. Adr_préc :=257

3) Fin tant que

Fin.

La chaine de sortie est : 'abababbaba'

6. Conclusion

Dans ce chapitre, nous avons d'abord présenté la méthode VLC. Puis, nous avons réalisé une analyse et une conception d'un système de compression / décompression LZW_VLC. Ensuite, nous avons présenté les différents modules principaux qui constituent l'application. Enfin, dans la conception nous avons spécifié d'une façon formelle les différents algorithmes et organigrammes qui composent ce système de compression / décompression.

Chapitre V

Implémentation

&

Evaluation

1. Introduction

Dans ce chapitre nous allons présenter la mise en œuvre de la méthode proposée, ainsi que les résultats des différents tests effectués dans le but d'évaluer les performances.

2. Environnement de développement

Le système d'exploitation choisi pour la réalisation de notre application est le système Windows. Ces systèmes sont très utilisés pour leurs simplicités, efficacités et fiabilités.

Nous avons choisi le langage C++ pour écrire notre programme, ce choix de langage est motivé par les raisons suivantes :

- Les compilateurs C++ sont actuellement implémentés sur toutes les plates-formes, ce qui fait du langage C++ un outil de programmation très répandu.
- Le code généré par le compilateur C++ est très optimisé, ce qui rend les exécutables plus compactes et plus rapides.
- La plupart des implémentations des algorithmes standards sont implémentés à base de langage C++.

Nous avons exploité l'environnement de développement DEV C++ pour la réalisation de cette application.

3. Implémentation

3.1. Interface de commande

L'exécution de l'application se fait sur l'invite de commande intégré sous les systèmes d'exploitation Windows et Linux (appelé Shell sous Linux).

Pour s'y faire il suffit de :

1. Atteindre le répertoire contenant les fichiers de l'application et les fichiers servants des tests.
2. Taper la commande suivant la syntaxe :
`LZW_VLC.exe <Option> <Fichier Entrée> <Fichier Sortie>`

- **Exemple**

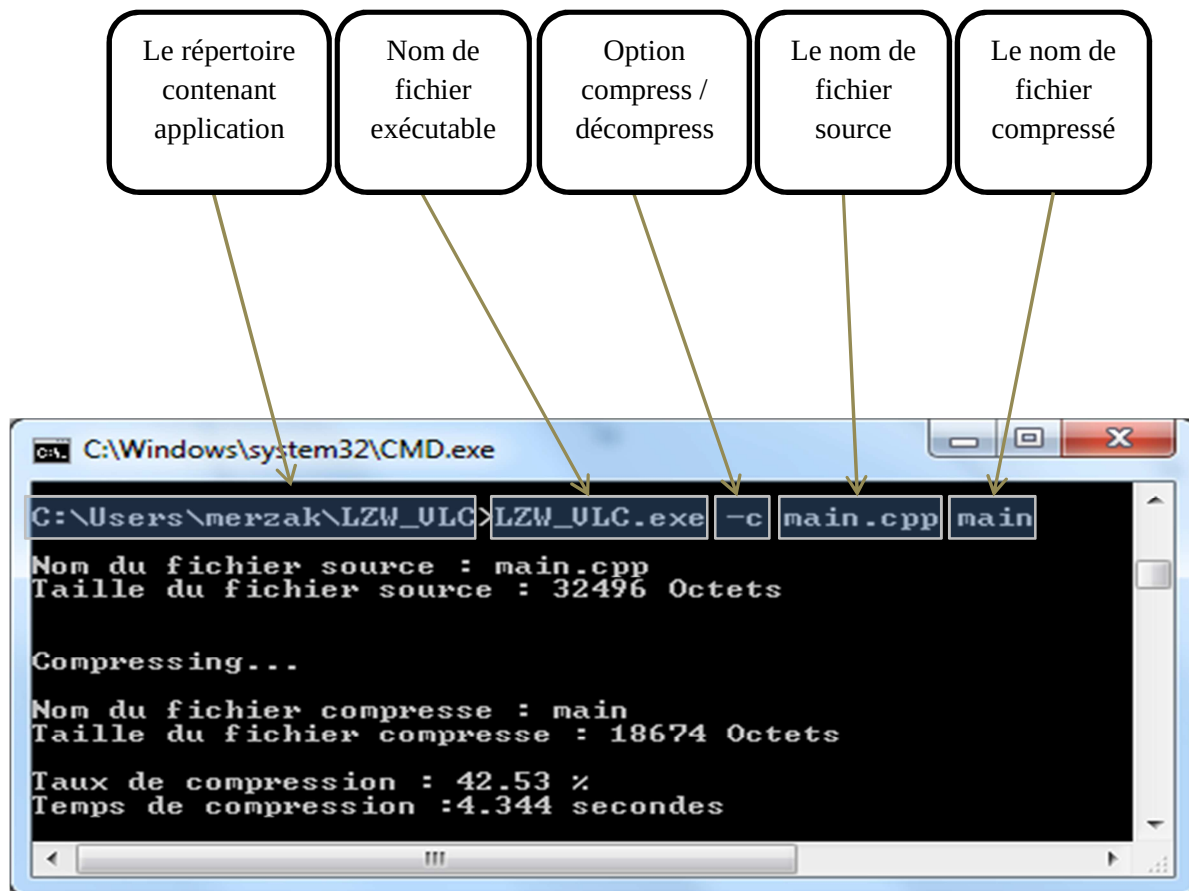


Figure V.1 : Exemple de compression avec LZW_VLC

3.2. Le noyau de l'application

Le noyau de notre application constitue de fonctions permettant d'englober les différentes actions afin d'implémenter les algorithmes présentés dans la partie conception (compression et décompression).

3.2.1. Les fonctions principales du noyau

➤ `MyCoder(in_name,out_name)`

Cette fonction récupère le nom du fichier à compresser et le nom du fichier sortie.

➤ `Coder()`

Cette fonction encode le fichier 'in_name' par le code LZW_VLC, et écrit les résultats dans le fichier 'out_name'

➤ MyDeCoder(in_name,out_name)

Cette fonction récupère le nom du fichier à décompresser et le nom du fichier sortie.

➤ Decoder()

Cette fonction décode le fichier 'in_name' par le code LZW_VLC, et écrit les résultats dans le fichier 'out_name'

4. Evaluation

Pour l'évaluation de temps et de taux de compression, les tests sont réalisés avec les trois méthodes : LZW, VLC et LZW_VLC sur les fichiers des corpus « CalgaryCorpus », « lukas_2d_8_tif » et « LargeCanterburyCorpus »

4.1. Présentation de taux et temps de compression

4.1.1. Taux de compression

1) Tests sur les fichiers de corpus « Lukas_2d_8_tif »

Les résultats des tests sur les fichiers de corpus « lukas_2d_8_tif » sont illustrés dans la figure V.2.

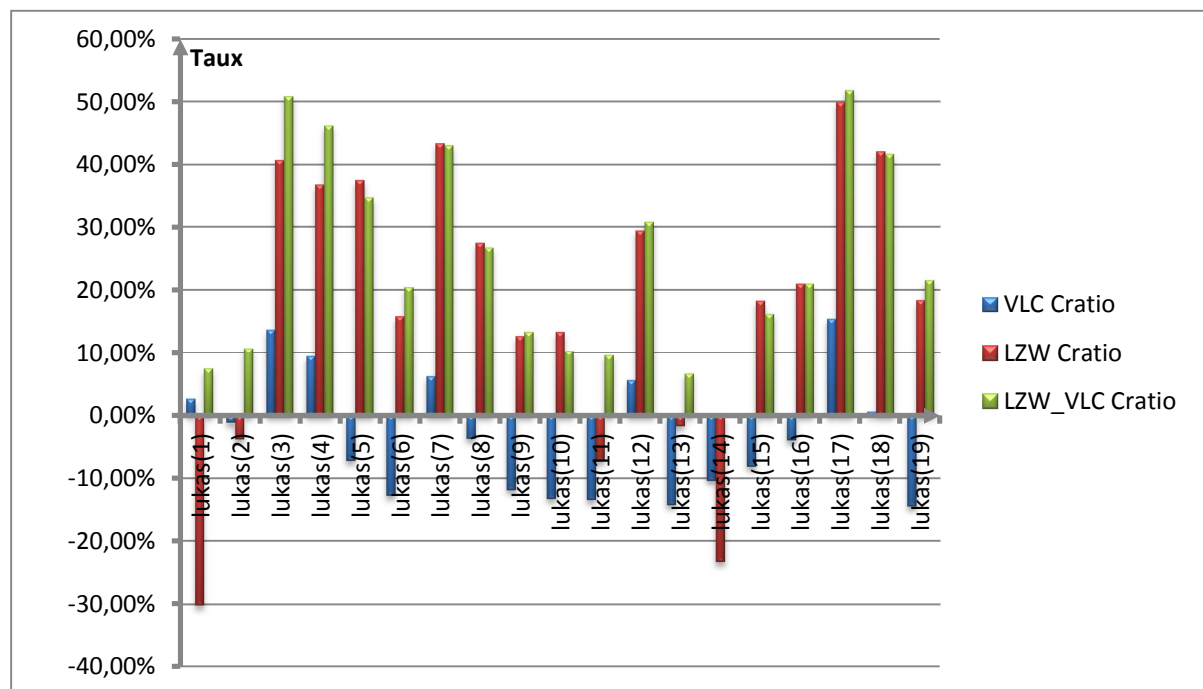


Figure V.2 : Graphe illustrant la différence de taux de compression entre VLC, LZW et LZW_VLC des tests réalisés sur « lukas_2d_8_tif »

La figure V.3 illustre la moyenne de taux de compression de chacune des trois méthodes sur les fichiers de corpus « lukas_2d_8_tif ».

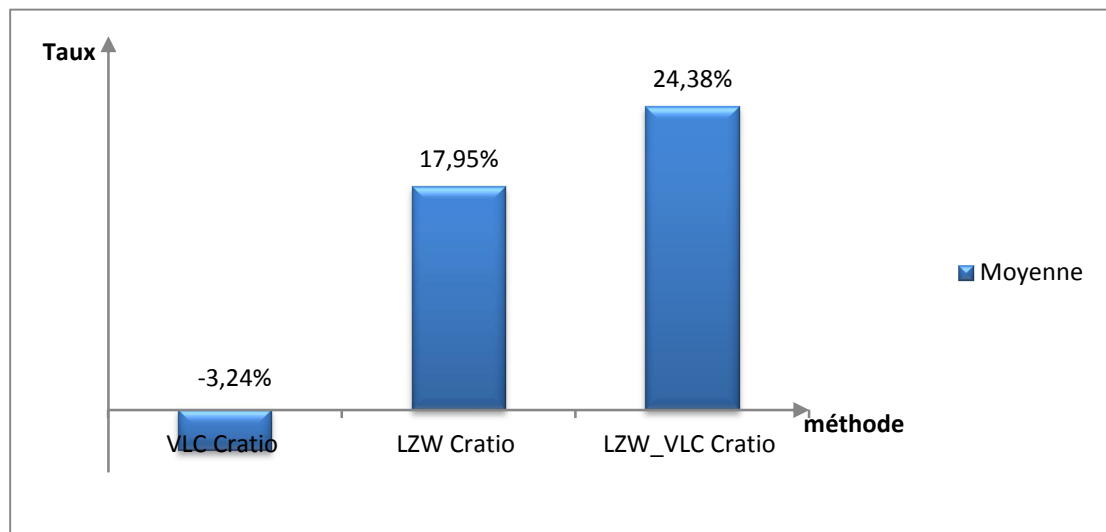


Figure V.3 : Graphe illustrant la différence de moyenne de taux de compression entre VLC, LZW et LZW_VLC des tests réalisés sur « lukas_2d_8_tif »

2) Tests sur les fichiers de corpus CalgaryCorpus

Les résultats des tests sur les fichiers de corpus « CalgaryCorpus » sont illustrés dans la figure V.4.

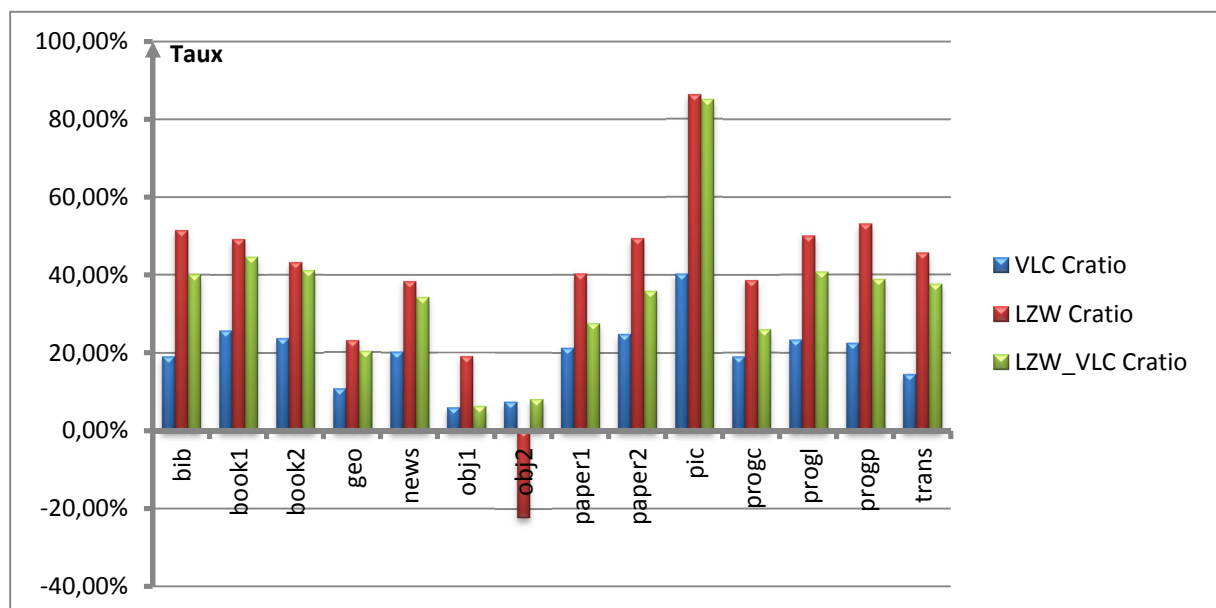


Figure V.4 : Graphe illustrant la différence de taux de compression entre VLC, LZW et LZW_VLC des tests réalisés sur « CalgaryCorpus »

La figure V.5 illustre la moyenne de taux de compression de chacune des trois méthodes sur les fichiers de corpus « CalgaryCorpus ».

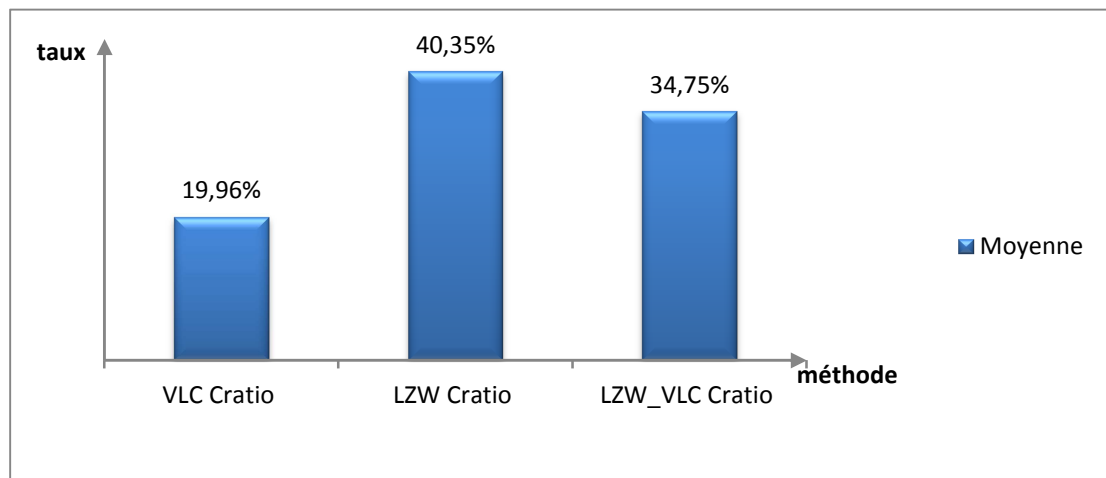


Figure V.5 : Graphe illustrant la différence de moyenne de taux de compression entre VLC, LZW et LZW_VLC des tests réalisés sur « CalgaryCorpus »

3) Tests sur les fichiers de corpus LargeCanterburyCorpus

Les résultats des tests sur les fichiers de corpus « LargeCanterburyCorpus » sont illustrés dans la figure V.6.

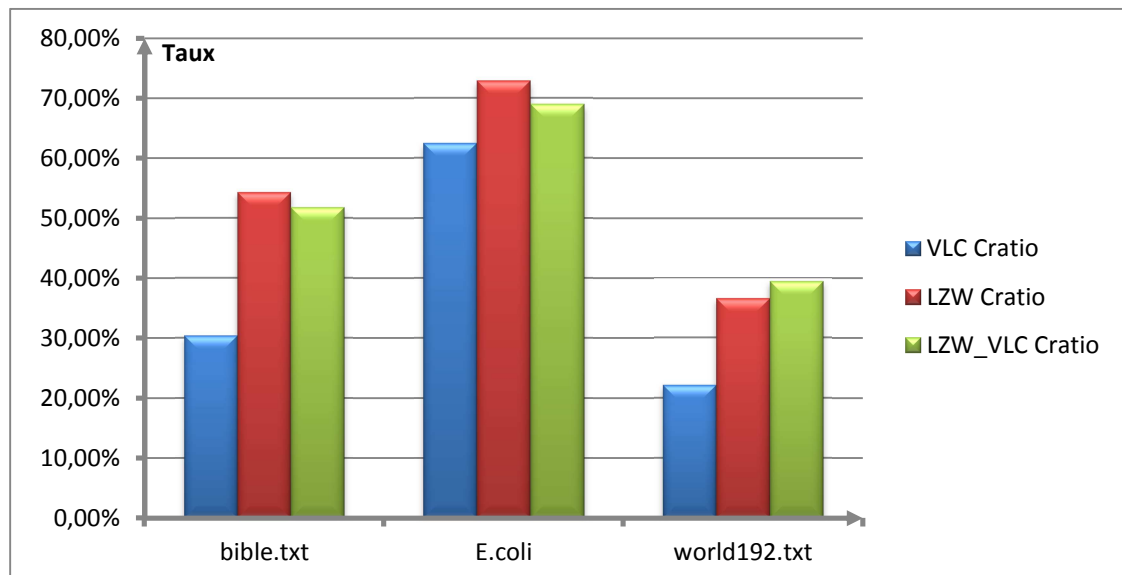


Figure V.6 : Graphe illustrant la différence de taux de compression entre VLC, LZW et LZW_VLC des tests réalisés sur « LargeCanterburyCorpus »

La figure V.7 illustre la moyenne de taux de compression de chacune des trois méthodes sur les fichiers de corpus « LargeCanterburyCorpus ».

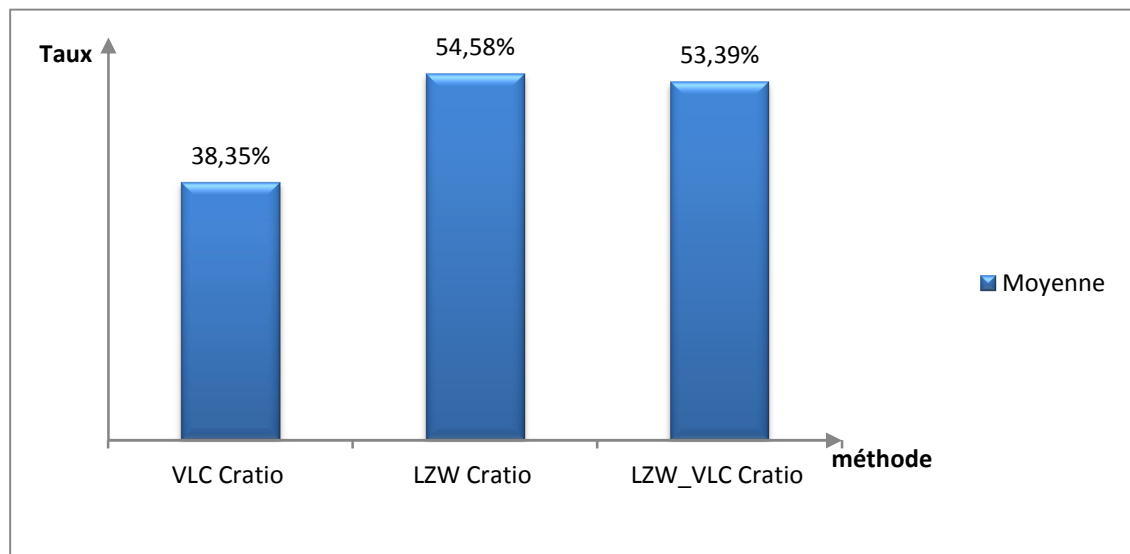


Figure V.7 : Graphe illustrant la différence de moyenne de taux de compression entre VLC, LZW et LZW_VLC des tests réalisés sur « LargeCanterburyCorpus »

4) Moyenne de taux de compression des trois méthodes sur les trois corpus

La figure V.8 illustre la moyenne de taux de compression de chacune des trois méthodes sur les fichiers des trois corpus.

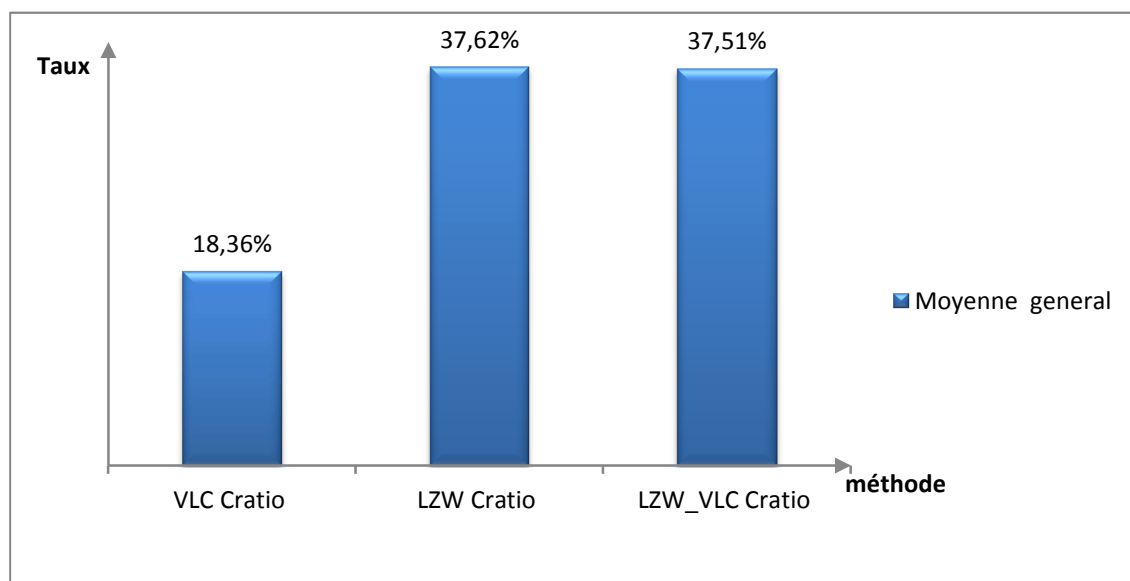


Figure V.8 : Graphe illustrant la différence de moyenne de taux de compression entre VLC, LZW et LZW_VLC des tests réalisés sur les trois corpus

4.1.2. Temps de compression

1) Tests sur les fichiers de corpus Lukas_2d_8_tif

Les résultats des tests sur les fichiers de corpus « lukas_2d_8_tif » sont illustrés dans la figure V.9.

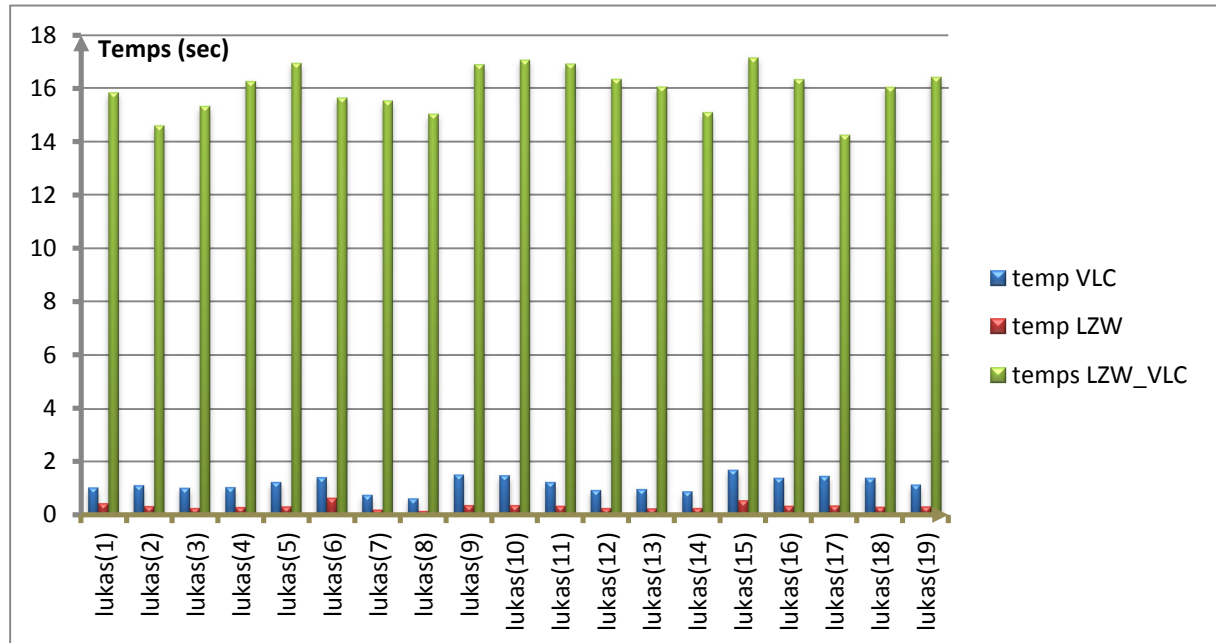


Figure V.9 : Graphe illustrant la différence de temps de compression entre VLC, LZW et LZW_VLC des tests réalisés sur « lukas_2d_8_tif »

La figure V.10 illustre la moyenne de temps de compression de chacune des trois méthodes sur les fichiers de corpus « lukas_2d_8_tif ».

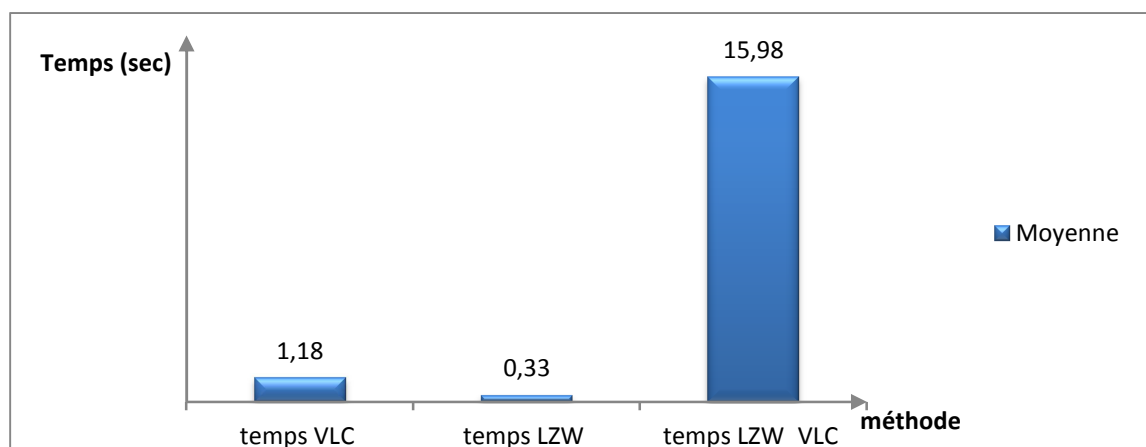


Figure V.10 : Graphe illustrant la différence de moyenne de temps de compression entre VLC, LZW et LZW_VLC des tests réalisés sur « lukas_2d_8_tif »

2) Tests sur les fichiers de corpus CalgaryCorpus

Les résultats des tests sur les fichiers de corpus « CalgaryCorpus » sont illustrés dans la figure V.11.

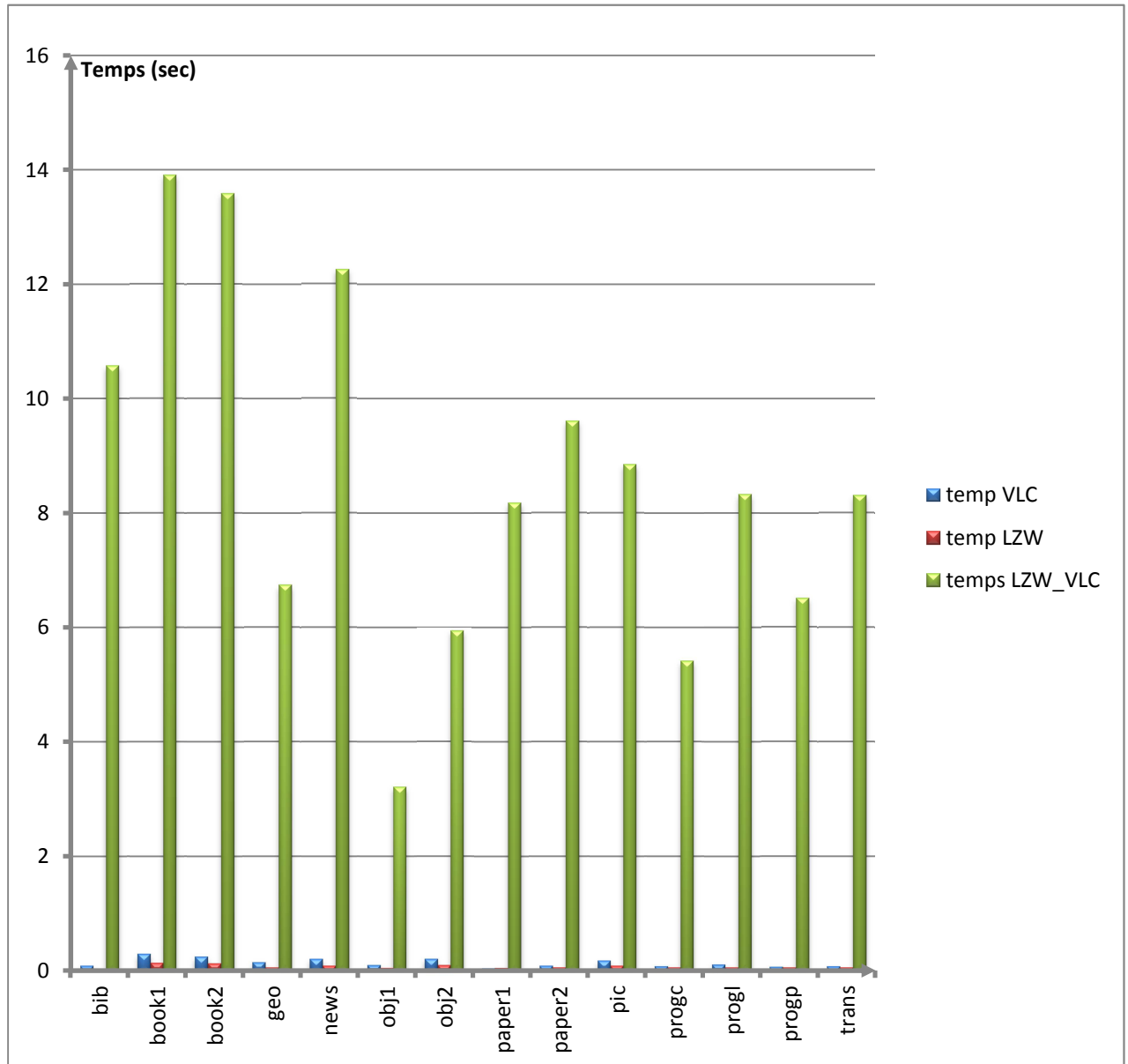


Figure V.11 : Graphe illustrant la différence de temps de compression entre VLC, LZW et LZW_VLC des tests réalisés sur « CalgaryCorpus »

La figure V.12 illustre la moyenne de temps de compression de chacune des trois méthodes sur les fichiers de corpus « CalgaryCorpus ».

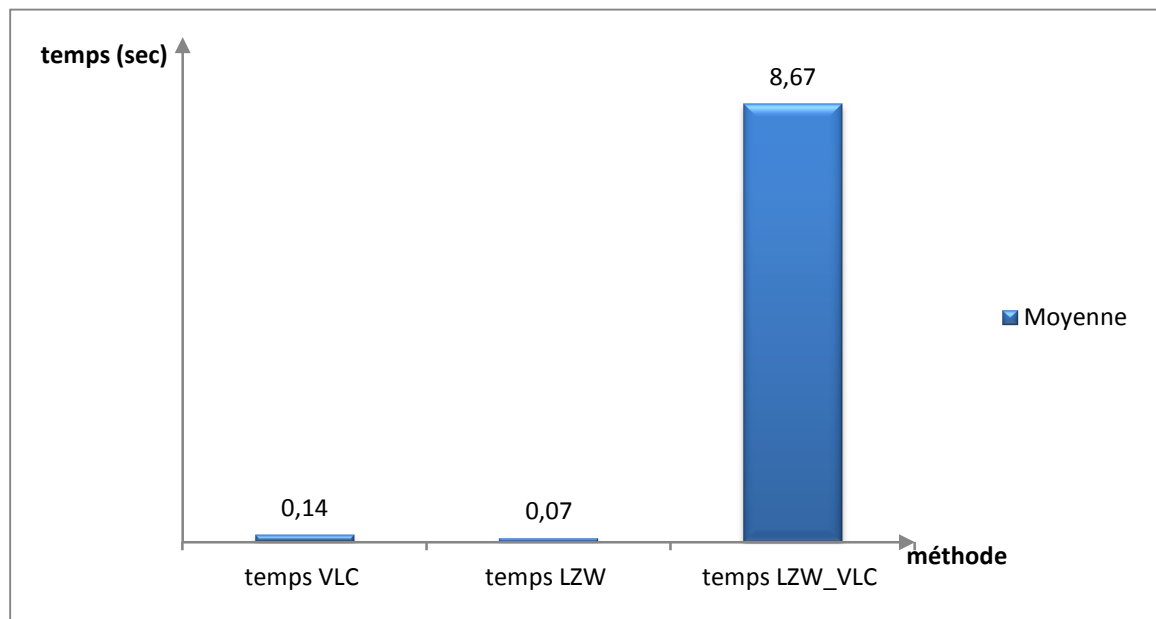


Figure V.12 : Graphe illustrant la différence de moyenne de temps de compression entre VLC, LZW et LZW_VLC des tests réalisés sur « CalgaryCorpus »

3) Tests sur les fichiers de corpus LargeCanterburyCorpus

Les résultats des tests sur les fichiers de corpus « LargeCanterburyCorpus » sont illustrés dans la figure V.13.

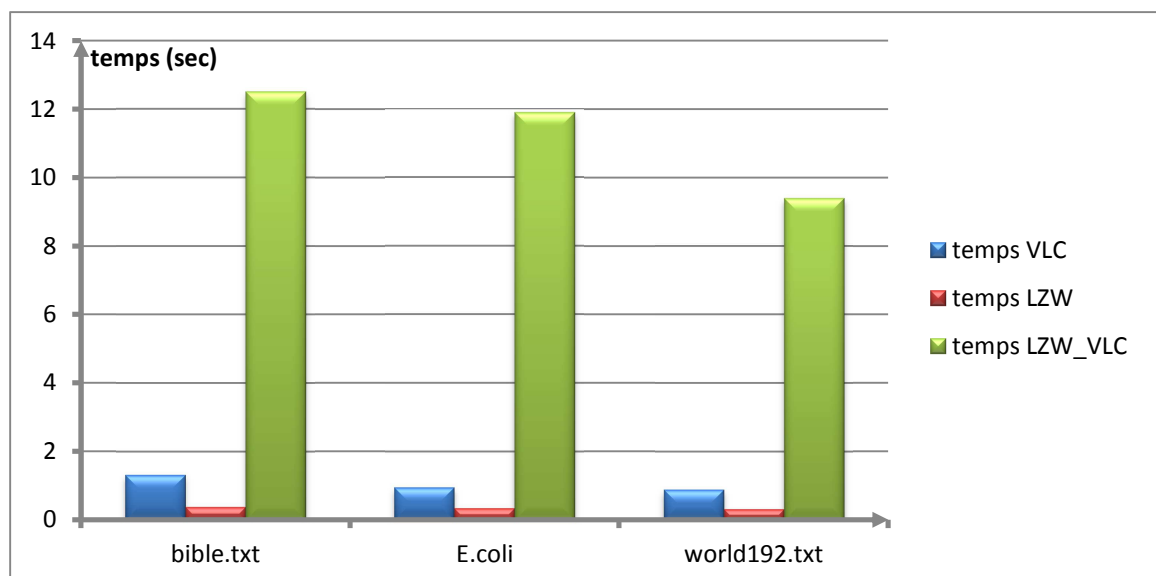


Figure V.13 : Graphe illustrant la différence de moyenne de temps de compression entre VLC, LZW et LZW_VLC des tests réalisés sur « LargeCanterburyCorpus »

La figure V.14 illustre la moyenne de temps de compression de chacune des trois méthodes sur les fichiers de corpus « LargeCanterburyCorpus ».

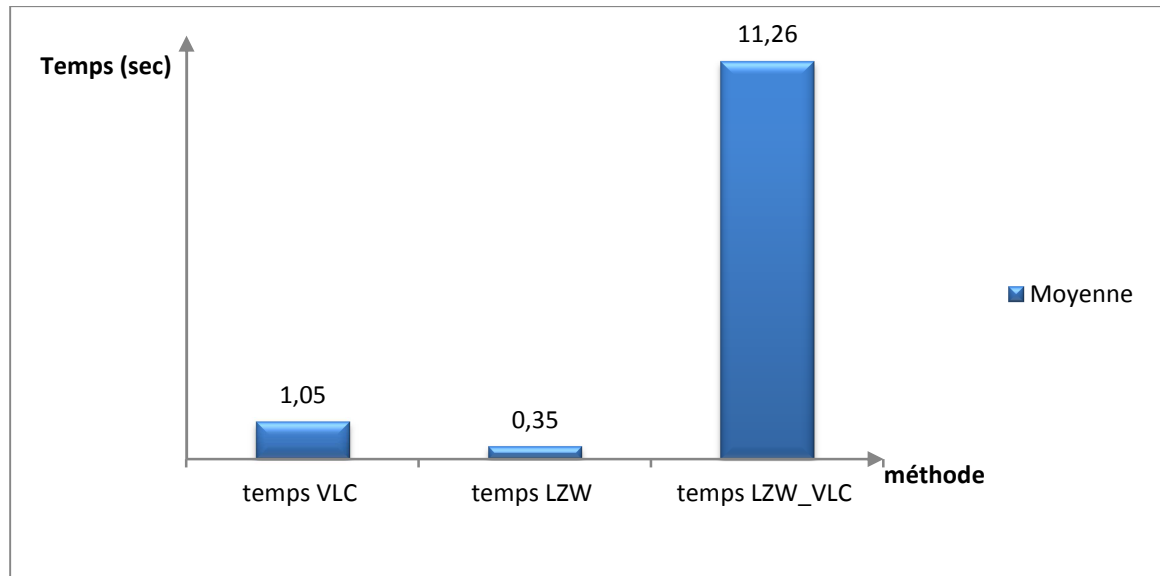


Figure V.14 : Graphe illustrant la différence de moyenne de temps de compression entre VLC, LZW et LZW_VLC des tests réalisés sur « LargeCanterburyCorpus »

4) Moyenne de temps de compression des trois méthodes sur les trois corpus

La figure V.15 illustre la moyenne de temps de compression de chacune des trois méthodes sur les fichiers des trois corpus.

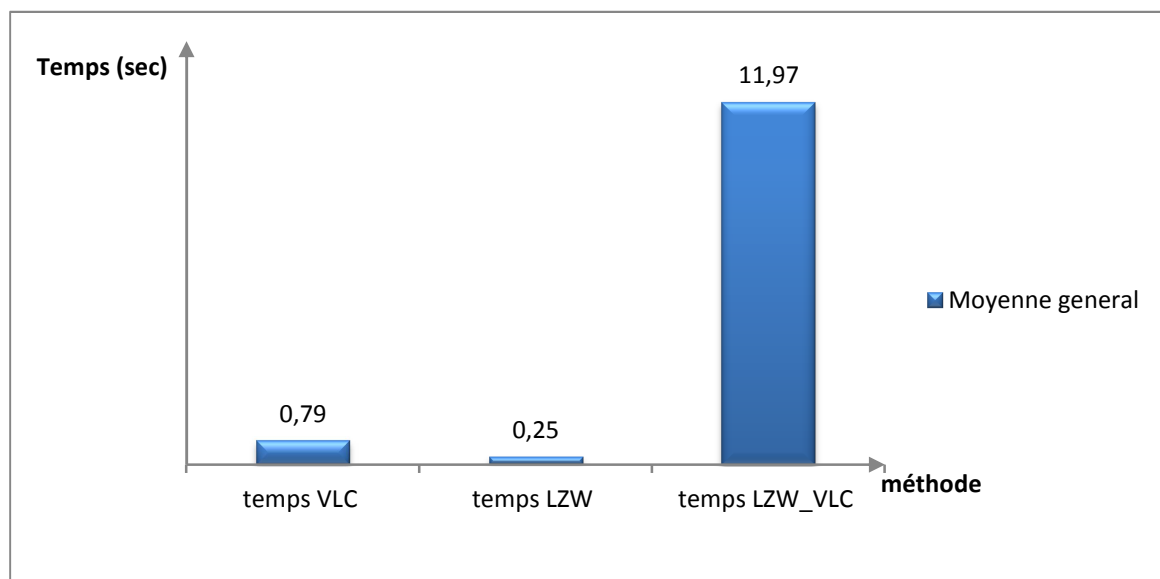


Figure V.15 : Graphe illustrant la différence de moyenne de taux de compression entre VLC, LZW et LZW_VLC des tests réalisés sur les trois corpus

4.2. Evaluation des résultats

4.2.1. Taux de compression

En observant d'une manière générale les taux des compressions obtenues des tests effectués sur différents corpus en utilisant les trois méthodes (LZW, VLC et LZW_VLC), on constate que le taux de compression de la méthode LZW_VLC et celui de la méthode sont très proche, tandis que le taux de compression de la méthode VLC est plus faible.

Mais en analysant de plus près, on voit clairement que :

- Sur les fichiers image, la nouvelle méthode LZW_VLC est plus efficace que LZW et VLC. La raison de cette efficacité est due au fait que les fichiers image sont des suites et répétitions de pixels et chaque pixel est représentés sur trois Octet (24 bits). Toute en sachant que la probabilité que LZW envoie un même pixel plus d'une fois n'est pas négligeable, Un pixel qui apparaît pour la deuxième fois ou plus sera écrit sur un nombre de bit inférieur ou égal à treize (13) bit en utilisant la nouvelle méthode, ce qui n'est pas le cas pour les méthodes LZW et VLC tel que la première l'écrit sur un nombre de bit fixe qui est douze (12) bit, tandis que la deuxième le traite comme une suite d'octet au lieu d'un bloc.
- Sur les fichiers texte, le taux de compression de la nouvelle méthode est meilleur que celui de VLC et faible par rapport à celui de LZW. Cette faiblesse par rapport à la méthode LZW est due au fait que la probabilité que la méthode LZW envoie le même code plus d'une fois est très faible et au fait que la nouvelle méthode a un entête de plus et un bit devant chaque nouveau code à écrire dans le fichier compressé.

4.2.2. Temps de compression

Concernant le temps de compression, cette méthode nécessite un temps beaucoup plus élevé que celui de LZW et VLC quel que soit le type de fichier compressé. Cela est dû au grand nombre d'instruction à exécuter.

5. Conclusion

Dans ce chapitre nous avons vus d'abord, les fonctions principales qui constituent l'application. Ensuite, nous avons effectué plusieurs tests sur différents types de fichier pour évaluer le taux et le temps de compression. Enfin, nous avons présenté les différents résultats

obtenus après ces tests avec une évaluation des résultats et des justifications pratiques et théoriques.

Les résultats obtenus montrent que cette nouvelle méthode couvre les faiblesses de compression de la méthode VLC sur les fichiers texte et améliore le taux de compression des deux méthodes (LZW et VLC) sur les fichiers image. Donc d'une vue globale, cette méthode est une amélioration pour VLC et une équilibrage pour LZW par rapport aux différents types de fichier.

Conclusion

Générale

La compression des données est un domaine très vaste qui sert à résoudre des problèmes de volume, de stockage et de débit. De nombreux travaux de recherche sont en cours afin d'améliorer les performances des techniques de compression.

Notre travail avait comme objectif l'implémentation et l'évaluation d'une nouvelle méthode de compression des données temps réel sans perte, basée sur les approches LZW et VLC. Pour cela il est nécessaire de comprendre divers concepts dans le monde de la compression des données et de la programmation.

A la fin de notre travail, nous avons constatés que cet algorithme de compression des données donne de meilleurs résultats de taux de compression pour les fichiers images mais pas pour les fichiers textes. Mais en gros, cette nouvelle méthode donne un taux de compression équivalent à celui donné par LZW, et surtout elle augmente l'efficacité des deux approches (LZW et VLC) sur les fichiers images.

Cette expérience nous a permis donc d'acquérir beaucoup de connaissances dans le domaine de compression des données, ainsi de mieux se familiariser avec le langage C / C++.

Le travail qu'on a fait n'a pas pu être complet, cela à cause des contraintes logiciels dont nous disposons, pour cela nous précisons que les différentes parties de notre application peuvent être maintenues et optimisées pour donner encore de bons et de meilleurs résultats.

- [1] : **Olivier RIOUL** : « Codage entropique à longueur variable » article 2003.
 - [2] : **Frederic KORICHE** « Cours Théorie de l'Information Compression : partie II », Université Montpellier II, France
 - [3] : **MM. S.Maadi, Y. Peneveyre, et C. Lambercy** « compression de donnée sans perte » article
 - [4] : **David SALOMON**: « A concise introduction to data compression», livre, 2007
 - [5]: **Vincent PEREIRA – Franck LEPRETTE - Vincent HACAULT**: « Compression de données » , Décembre 2004
 - [6] : **Abdelhalim GABIS** : « Compression et Segmentation des Images », Thèse de magistère, INI 2005.
 - [7] : **M. Thiebert & M. Jussiaux** :« Les méthodes de compression graphiques et les formats informatiques concernés »
 - [8] : **Mark Nelson** : « La Compression de données » , DUNOD, Paris 1993. Traduit par Hervé Soulard de « The Data Compression Book » California 1992.
 - [9]: **Darrel HANKERSON , Greg A. HARRIS , Peter D. JOHNSON**: Livre «Information Theory and Data Compression», second edition 2003.
 - [10]: **Khalid SAYOOD** :« lossless compression handbook » 2003
 - [11]: **Peter FENWICK** : « punctured Elias codes for variables-length coding of the integers», article ,december 5, 1996
 - [12] : **David SALOMON** « variable-length code for data compression» , livre.
 - [13] : **David SALOMON** « data compression» fourth edition, Livre.
 - [14] : **Vincent BEAUDOIN** « Developpement de nouvelles techniques de compression de donnees sans perte » Memoire presente a la Faculte des etudes superieures de l'Universite Laval (Québec) dans le cadre du programme de maitrise Informatique pour l'obtention du grade de maitre en sciences (M.Sc.)
 - [15] : **Denis KUBASOV** : « Codage de sources distribuées : nouveaux outils et application à la compression vidéo » thèse de doctorat ,2008
-