République Algérienne Démocratique et Populaire
Ministère de l'Enseignement Supérieur et de la Recherche Scientifique
Université Mouloud Mammeri, Tizi-Ouzou
Faculté de Génie Electrique et d'Informatique

Département Informatique.

MEMOIRE DE MASTER

En Informatique

Option : Conduite d'un projet informatique

Thème :

Compression temps réel des pages web

Proposé et dirigé par :

M^r^ : SADOU Samir

devant le jury :

Président :

Réalisé par :

M^elle :^ ISSAAD Kahina

M^elle :^ KHOUDI Hayet

Examinateurs :

Promotion : 2012

Remerciements

Nous remercions le tout puissant, qui nous a donné la force et la patience pour l'accompli cément de ce travail.

Nous remercions chaleureusement notre promoteur, Mr SADOU. Samir pour nous avoir proposé ce sujet , en nous faisant confiance , ainsi que pour avoir dirigé notre travail avec ses orientations ,ses précieux conseils , ses remarques constructives et surtout son sérieux et sa disponibilité .

Un grand merci aux membres de jury pour l'honneur qu'ils nous ont attribué pour évaluer et juger notre travail.

Merci à nos familles et amis pour leurs soutiens et leurs encouragements.

Et à tous ceux qui de prés ou de loin nous ont aidés à l'aboutissement de cette quête.

Merci infiniment à tous.

Kahina et hayet

DÉDICACES

A mes très chers parents

A la mémoire de mes grands parents

A mes fréres samil ,djallal , anis et ma sœur sarah

A ma binôme hayet et toute sa famille

A mes cousins et cousines

A toute ma famille

A mes amies

A tous qui ceux m'ont aidé de prés ou de loin dans mes études.

-Kahina –

DÉDICACES

A mes très chers parents

A mes très chers grands parents

A mes sœurs

A mes très chers oncles et tentes.

A ma binôme kahina et toute sa famille

A toute ma famille

A mes amies

A tous qui ceux m'ont aidé de prés ou de loin dans mes études.

- Hayet-

Liste des figures

Chapitre I : généralités sur la compression de données

Chapitre II : Les méthodes VLC.

Liste des figures

Chapitre III : Analyse et conception.

Chapitre IV : Implémentation et évaluation.

Liste des tableaux

Chapitre I : généralités sur la compression de données.

Chapitre II : Les méthodes VLC.

Chapitre III : Analyse et conception.

Sommaire

----------Chapitre 2 : Les méthodes VLC----------

----------Chapitre 3 : Analyse et conception----------

Sommaire

Grâce aux réseaux et à Internet particulièrement, la quantité d'information échangée entre les usagers a considérablement augmenté ,Ceci nécessite des supports de stockage puissants et performants pour satisfaire la forte demande de l'espace de stockage, et pour garantir un minimum de trafic sur un réseau qui souffre de bande passante limitée

Une solution très courante est de minimiser la taille de la représentation de l'information, c'est le problème de la compression de données.

La compression de données est une branche importante de la théorie de l'information, elle est utilisée pour réduire la taille physique d'un bloc d'information, c'est-à-dire réduire la longueur d'une chaine sans affecter son contenu informatif. En compressant des données, on peut placer plus d'information dans le même espace de stockage, ou utiliser moins de temps pour le transfert au travers d'un réseau téléinformatique. C'est pour cela que les données ont souvent besoin d'êtres compressés.

Vu l'importance de ce domaine, plusieurs études et recherches ont été faites, et Plusieurs méthodes et algorithmes de compression ont vu le jour. Cette variété de méthodes est due à la diversité des types de données ciblés : images, audio, vidéo, texte, etc.

L'éventail des algorithmes de compression de données est immense. Deux grands types d'algorithmes sont à distinguer :

- ✓ **Les algorithmes de compression sans perte** : c'est-à-dire lors de la décompression du fichier compressé nous obtenons le même fichier d'origine, ces algorithmes sont utilisé généralement Pour les fichiers de données ou les programmes, car lors de la décompression on doit retrouver le fichier dans son état d'origine, à l'octet près.

- ✓ **Les algorithmes de compression avec perte** : ils permettent a l'utilisateur-cible de recouvrer intégralement les données, ces algorithmes sont utiliser principalement pour d'autres types de données, comme le son et les images, il est envisageable d'accepter une perte de données, c'est-à-dire une baisse de qualité du son ou de l'image, pour bénéficier en contrepartie d'un volume de données réduit, plus facile à stocker et plus facile à transmettre.

La méthode étudiée dans notre travail est une méthode de compression sans perte, qui se base sur le codage à longueur variable (VLC).

Pour mener à terme notre travail, nous le répartissons sur quatre chapitres :

- ✓ **Chapitre I :** dans ce chapitre on rappelle les concepts de base de la compression des données avec des définitions et des illustrations.
- ✓ **Chapitre II :** ce chapitre est consacré pour la présentation de la compression à base des codes VLC (Variable Length Coding), ainsi quelques algorithmes qui utilisent ce type de codage.
- ✓ **Chapitre III :** dans ce chapitre nous avons présenté en premier lieu le principe et le fonctionnement de la méthode proposée. Ensuite nous avons fait la conception en présentant les différents modules.
- ✓ **Chapitre IV :** ce chapitre présente l'environnement de développement et les principales fonctions qui constituent notre application, ainsi que les résultats de compression obtenus.

Enfin nous avons terminé par une conclusion qui montre les rapports de ce travail.

CHAPITRE 1 :

GÉNÉRALITÉS SUR LA COMPRESSION DE DONNÉES

I. Introduction

Les données informatiques utilisent un espace disque non négligeable, de même lorsqu'on souhaite transmettre des données (flux vidéo, sonore…) sur le réseau internet, la bande passante allouée pour le transfert est très limitée en terme de débit et de capacité de gestion de gigantesque volume de paquets de données, ce qui diminue considérablement la qualité du service internet , C'est pour ces raisons que la compression de données est presque toujours utilisée, pour réduire autant que possible la quantité de données à stocker ou à transmettre.

II. La Théorie de l'information [1] [16]

La théorie de l'information est due à Shannon (vers 1948), avec l'influence des grands théoriciens de l'informatique (Turing, Von Neumann, Wiener). Elle est la première grande théorie systématique (et même mathématique) de la communication. Elle définit celle-ci comme un transfert d'information entre un émetteur et un récepteur à travers un canal de communication. Cette information ne peut être véhiculée que sous la forme d'un code commun aux deux parties. Les défauts de transmission de l'information par le canal sont appelés bruit.

La théorie de l'information fournit une mesure quantitative de la notion d'information apportée par un message (ou une observation). Elle s'appuie non seulement sur les communications mais aussi sur l'informatique et la statistique.

II. 1. Le codage [2]

Dans un système de communication, des messages différents portent la même information, le codage cherche le message avec les meilleures propriétés.

- **Le codage de source** : supprime la redondance, réduit le cout.
- **Le codage de canal** : protège contre les perturbations.
- **Le cryptage** : destiner à sécuriser les informations.

II. 2. Concepts de la théorie de l'information [3]

La théorie de l'information a prie sa dimension avec l'élaboration de sa théorie mathématique par SHANON et WEAVER en définissant les concepts de quantité d'information, entropie et redondance.

II. 2.1. Quantité d'information [34]

La définition mathématique de la quantité d'information repose sur une théorie probabiliste. La quantité d'information contenue dans un flot de données est liée à sa probabilité d'apparition.

Soit *S*, une source d'information (alphabet) $S=(X_1, X_2, X_3, \ldots\ldots, X_k)$ de *K* symboles. Et soit $P(X_i)$ la probabilité d'apparition deX_i.

La quantité d'information (L_i) contenue dans un flot de données est une fonction de l'inverse de la probabilité d'apparition de ce symbole, elle est donnée par la relation suivante:

$$L_i = \log_2\left({}^{1}/_{P(X_i)}\right) = -\log_2(P(X_i))$$

L'unité de la quantité d'information est le Shannon.

II. 2.2. Entropie [4]

C'est une notion empruntée par la thermodynamique, En théorie d'information, l'entropie mesure le degré du hasard dans un flot de données (fichier).Un fichier qui a un faible taux d'entropie, dont les éléments sont tous prévisibles à l'avance, sera beaucoup compactable.

Par exemple un fichier rempli de zéros. La quantité d'information qu'il contient est nulle, le hasard inexistant, l'entropie minimale. Au contraire un fichier zippé (déjà compacté) contient des octets apparemment aléatoires sans aucune relation les uns avec les autres. La quantité d'information y est maximale, presque par définition puisqu'il s'agit d'un fichier déjà compressé, donc dont on a à priori supprimé toutes les redondances. Il contient par ailleurs une dose de hasard maximale, soit une entropie maximale.

L'entropie H d'un fichier est l'information moyenne contenue par chaque symbole, elle est donnée par la relation suivante.

$$H = \sum_{i}^{n} L_i\, P(X_i) = -\sum_{i}^{n} P(X_i)\ \log_2(P(X_i))$$

Avec n : le nombre maximum de symboles (Si) existant.

- ✓ L'unité de l'entropie est le bit (de binary) ou le Shannon(Sh).
- ✓ L'entropie représente aussi le nombre de bits par symbole qui sont nécessaire pour coder ce symbole. Ce nombre de bits est la longueur L du code L≥H.

- ✓ **Propriétés**
  - $0 \leq H \leq \log_2(n)$.
  - $H = 0$ lorsque $P(X_i) = 1\ et\ P(X_j) = 0, i \neq j$.

- $H = \log_2(n)$ lorsque $P(X_i) = {}^1/_n$, $\forall\, i$.

II. 2.3. La quantité de décision [17]

D est liée au nombre de symboles N parmi lesquelles la source choisie aux qu'elle va émettre.

D= log2 (N) ou N : le nombre de symboles.

II. 2.4. La redondance [3] [17]

Mesure la quantité d'information significative dans le débit de décision généré par une source, il est définit par :

R=D-H

La compression de données se base sur la détection et l'élimination des redondances dans un flot de données. Et sachant que l'information informatique quel que soit son format (texte, son, image, ….) est logiquement représenté par une combinaison de 256 octets différents (si on considère l'octet comme unité de base) appelé alphabet ; cela implique que chaque flot de données qui a une taille supérieure a celle de l'alphabet (nombre de symboles) présente nécessairement une redondance.

✓ **Exemple:** Soit *A* l'alphabet et *F* le flot de données.

A= {0,1} de taille 2 symboles, F=1001…10.F présente la redondance des symboles 0 et 1 dès que F dépasse la taille de 2. De ce fait, on peut dire que les flots de données de grande taille (plusieurs Mo) ne sont que le résultat de la redondance des symboles de l'alphabet qui est un ensemble borné.

✓ **Types de redondances [5] [18]**

Il ya plusieurs types de redondances qui peuvent exister dans un même flot de données :

➢ **Redondance de caractère**

Dans un flot de données, certains caractères peuvent avoir une fréquence d'apparition plus élevée que d'autre.il est donc possible de tirer profit de cette caractéristique pour réaliser une compression en agissant des codes courts aux caractères les plus fréquents et les codes longs aux caractères plus rares.

➢ **Redondance de groupe de caractères**

Dans l'alphabet de la langue anglaise par exemple le nombre de combinaisons de caractères qui forme un mot est inférieure au nombre de combinaisons possibles, comme les mots « the », « are », « what », « was », peuvent apparaître dans un texte par contre il est presque

impossible de rencontrer « zzzzdddd », donc on va associer au groupe de symboles le plus fréquent un code cours et au groupe de symboles rares un code long.

- **Corrélation entre les symboles**

Dans un flot De données, l'apparition d'un symbole peut augmenter la probabilité d'apparition d'un autre symbole après le premier.

Par exemple lorsqu'on rencontre 'T' dans un texte de la langue anglaise, il est fort probable que la lettre suivante serait un 'H' a cause de la forte corrélation qui existe entre ces deux lettres. en terme de probabilité, la phrase précédente peut s'écrire :P=(H/T) ≈1 qui se lit : « la probabilité d'avoir un H sachant qu'on a déjà un T est proche de 1 ». ainsi ,il est intéressant d'affecter des codes a des groupes de symboles fortement corrélés.

III. Le Codage

III. 1. définition [11]

Il s'agit d'une règle permettant de convertir une information sous une forme différente de sa représentation initiale.

- ✓ Ainsi, on peut scinder les « codes » en plusieurs grandes familles :
  - Les codes de communication (Morse, Baudot…..).
  - Les codes de représentation (ASCII, Base64,…).
  - Les codes de protection (cryptographie,….).
  - Les codes de compression (le codage VLC,……).
  - Les codes d'identification (Code-barres,…..).

III. 2. La propriété d'un code

Pour avoir un codage correct, il faut qu'il vérifie les propriétés suivantes :

- ✓ Tous les mots du code peuvent êtres distingués.
- ✓ Le décodage ne donne lieu à aucune ambigüité.

III. 3. Le code préfixé [18]

On dit qu'un code « c » sur un vocabulaire « v » est a la propriété du préfixe si et seulement si pour tout couple de mots de code distincts (c1, c2), c2 n'est pas un préfixe de c1.

Exemple : A=101000, B=01, C=1010 : B n'est pas un préfixe de A mais C est un préfixe de A. Grace a la propriété du préfixe, on peut déchiffrer les mots d'un tel code dès la fin de la réception du mot.

- ✓ Tout code possédant la propriété du préfixe est uniquement déchiffrable.
- ✓ Tout code dont tous les mots sont de même longueur possède la propriété du préfixe.

IV. La compression des données

IV. 1. Définition [20]:

La compression de données ou codage de source est l'opération informatique qui consiste à transformer une suite de bits A en une suite de bits B plus courte, en utilisant un algorithme particulier.

Il s'agit d'une opération de codage, c'est-à-dire changer la représentation de l'information, dans le but de rendre la représentation compressée plus courte que la représentation originale. La **décompression** est l'opération inverse de la compression.

- ✓ La théorie de la compression de données est issue de la théorie de l'information.

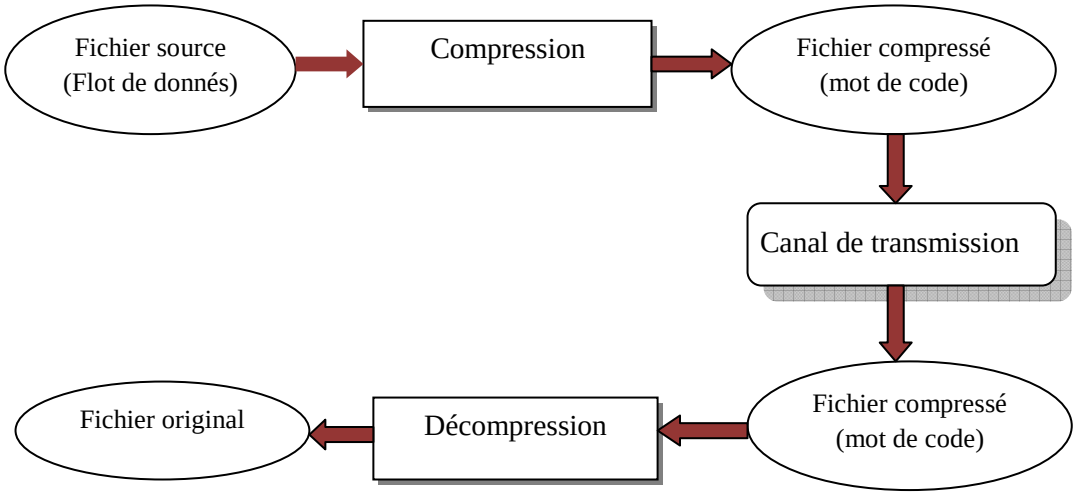


Figure 1.1 principe de compression/décompression.

IV. 2. Estimation du taux de compression

Il s'agit du pourcentage de compression appliqué à un fichier. Pour la compression avec perte, le taux de compression doit être choisi avec attention pour ne pas rendre le fichier inutilisable. En revanche, la compression sans perte peut supporter des taux de compression très élevés. Il faut également noter que plus le taux est élevé, plus il faut de temps pour les opérations de compression et de décompression.

$$\textbf{Taux de compression} = \frac{(taille\ originale - taille\ finale) * 100}{taille\ original}$$

V. Catégorisation de compression : [6] [11] [19]

Il existe plusieurs manières de classer les formats de compression. En voici quelques exemples :

- Par analyse statistique ou par substitution.
- Avec ou sans perte d'information.
- Symétrique ou asymétrique.
- Compression physique/logique.

V. 1. La Compression par codage statistique/substitution

V. 1.1 Par codage statistique [11]

Le codage statistique a pour but de Réduire le nombre de bits utilisés pour le codage des caractères fréquents, et augmenter ce nombre pour des caractères plus rares (exemple : codage de Huffman, Codage de Shannon-Fano).

V. 1.1.1 L'algorithme de Huffman [12]

Le codage de Huffman est un algorithme de compression des données sans perte basé sur les fréquences d'apparition des caractères apparaissant dans le document initial. Il a été développé par un étudiant de la MIT, David Albert Huffman en 1952.

Cette technique est largement utilisée car elle est très efficace et nous observons selon le type de données des taux de compression allant de 20% à 90% mais plus généralement entre 30% et 60%.

C'est un codage entropique de type VLC (codage à longueur variable) qui consiste à coder les caractères qui apparaissent souvent dans un texte par un code binaire court, et ceux qui apparaissent plus rarement par un code plus long. Par exemple la lettre « e » est très utilisée dans la langue française, tandis que le « z » beaucoup moins.

- **Algorithme de compression :**

1. on cherche la fréquence des caractères.
2. on trie les caractères par ordre décroissant de fréquence.
3. on construit un arbre pour donner le code binaire de chaque caractère

✓ **Construction de l'arbre**

On relie deux à deux les caractères de fréquence les plus basses et on affecte à ce nœud la somme des fréquences des caractères. Puis on répète ceci jusqu'à ce que l'arbre relie toutes les lettres. L'arbre étant construit, on met un 1 sur la branche à droite du nœud et un 0 sur celle de gauche.

- **Exemple** : Algorithme de Huffman

Pour illustrer cet algorithme, nous allons coder la phrase suivante : "Le codage est indispensable".

Pour simplifier nous n'allons pas prendre en compte le symbole espace (blanc). Cette phrase est une source de 24 symboles. Tous ces symboles émanent de l'alphabet A.

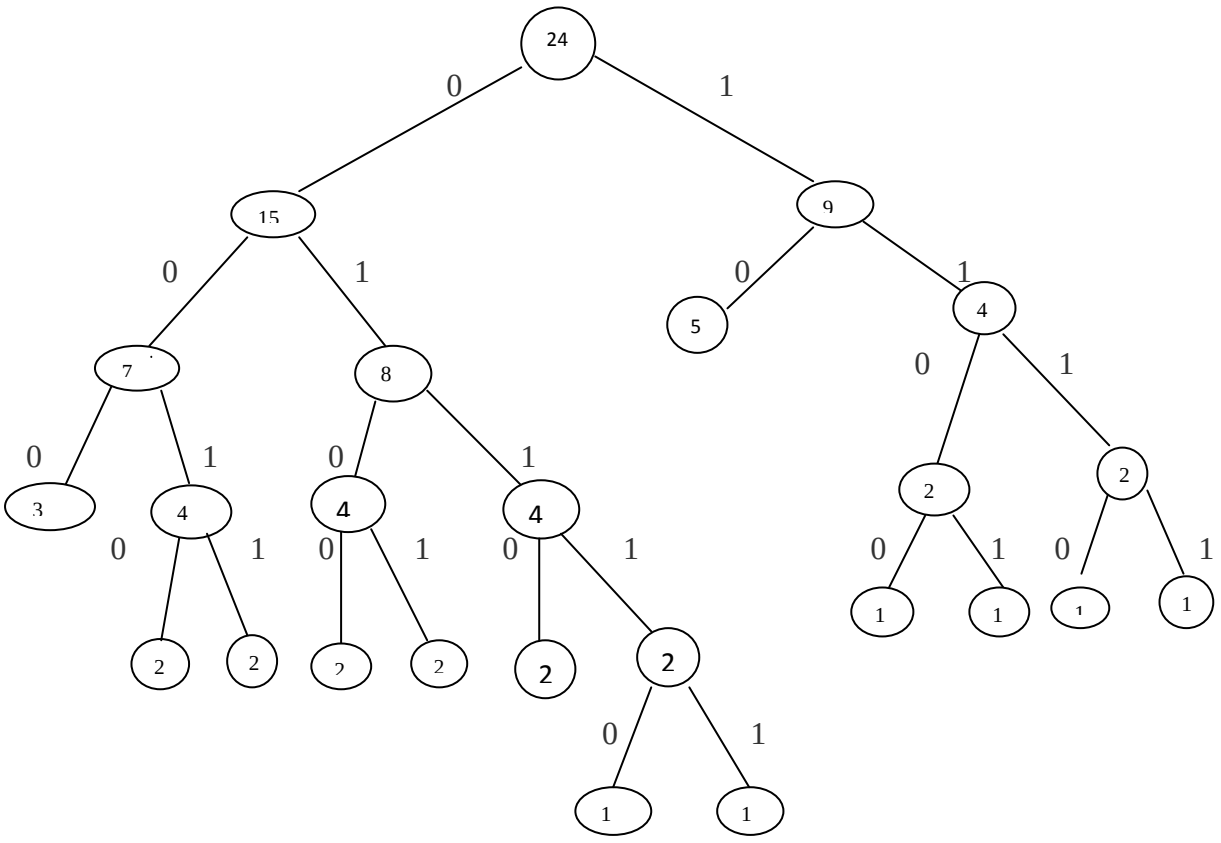
A={E , S , A ,D , I , N , L , B ,G , P ,T ,O ,C}, Cet Alphabet a N=13 symboles

✓ Le tableau des probabilités d'occurrence est le suivant :

symbole	**Nombre de fois**	**Probabilités**
E	5	5/24
S	3	3/24
A	2	2/24
D	2	2/24
I	2	2/24
N	2	2/24
L	2	2/24
B	1	1/24
G	1	1/24
P	1	1/24
T	1	1/24
O	1	1/24
C	1	1/24

Tableau 1.1 les probabilités d'occurrences pour la méthode de Huffman.

- ✓ On applique successivement les étapes de l'algorithme selon le principe de l'algorithme précédemment indiqué. L'arbre de Huffman associé est présenté ci-après.



S - A - D - I - N - L - B - G - E - P - T - O - C

000 - 0010 - 0011 - 0100- 0101 – 0110 - 01110 – 01111 – 10 - 1100 – 1101 – 1110 - 1111

Figure 1.2 Arbre de Huffman.

- ✓ Le fichier compressé se compose d'une suite de codes sans séparateur, bien que les codes comportent un nombre variable de bits, car chaque code à la propriété d'unicité de son préfixe.

- **Algorithme de décompression :**

On transmet la bibliothèque de codage, puis on associe les caractères à leur code.

- **Utilité et caractéristiques de compression :**

Cet algorithme permet de compresser aussi bien des images que des textes, on parle de compression statistique ou codage entropique (probabilités d'apparition d'un caractère).

On obtient une compression satisfaisante (50% en moyenne) et un temps de compression assez rapide. En revanche il faut transmettre la bibliothèque en même temps, et il arrive que la taille du fichier soit plus grande que celle du fichier non compressé. De plus il est très sensible à la perte d'un bit, toutes les valeurs qui suivront ce bit seront fausses lors de la décompression.

✓ Il existe trois variantes de l'algorithme de Huffman, chacune d'elle définit une méthode pour la création de l'arbre :

a- Statique

Chaque octet a un code prédéfini par le logiciel. L'arbre n'a pas besoin d'être transmis, mais la compression ne peut s'effectuer que sur un seul type de fichier (ex: un texte en français, où les fréquences d'apparition du 'e' sont énormes ; celui-ci aura donc un code très court).

b- Semi-adaptatif

Le fichier est d'abord lu, de manière à calculer les occurrences de chaque octet, puis l'arbre est construit à partir des poids de chaque octet. Cet arbre restera le même jusqu'à la fin de la compression. Il sera nécessaire pour la décompression de transmettre l'arbre.

c- Adaptatif

C'est la méthode qui offre a priori les meilleurs taux de compression car l'arbre est construit de manière dynamique au fur et à mesure de la compression du flux. Cette méthode représente cependant le gros désavantage de devoir modifier souvent l'arbre, ce qui implique un temps d'exécution plus long. Par contre la compression est toujours optimale et le fichier ne doit pas être connu **avant** de compresser. Il ne faut donc pas transmettre ou stocker la table des fréquences des symboles. De plus, l'algorithme est capable de travailler sur des flux de données (*streaming*), car il n'est pas nécessaire de connaître les symboles à venir.

V. 1.1.2 L'algorithme de Shannon-Fano [12]

Ce procédé est antérieur au codage de Huffman est se base également sur un codage statistique.

- **Algorithme**

1. Construire une table des fréquences d'apparition des symboles triée par ordre décroissant.
2. Diviser cette table en deux parties. Celles-ci doivent avoir une somme de fréquences égale (ou pratiquement égale) à celle de l'autre.
3. Affecter le chiffre 0 à la moitié inférieure, la moitié supérieure prenant la valeur 1.
4. Répéter les opérations 2 et 3 aux deux parties, jusqu'à ce que chaque symbole ne représente plus qu'une partie de la table.

- **Exemple** : Application de l'algorithme de Shannon-Fano

Pour illustrer cet algorithme, nous allons coder la phrase suivante : "Le codage est indispensable".

Pour simplifier nous n'allons pas prendre en compte le symbole espace (blanc). Cette phase est une source de 24 symboles.

Tous ces symboles émanent de l'alphabet A={E , S , A ,D , I , N , L , B ,G , P ,T ,O ,C}, Cet Alphabet a N=13 symboles.

✓ **ETAPE 1**

symbole	**Nombre de fois**	**Probabilités**
E	5	5/24
S	3	3/24
A	2	2/24
D	2	2/24
I	2	2/24
N	2	2/24
L	2	2/24
B	1	1/24
G	1	1/24
P	1	1/24
T	1	1/24

O	1	1/24
C	1	1/24

Tableau 1.2. Etape 1 de l'algorithme de Shannon-Fano.

Application successive des étapes 2,3 et 4

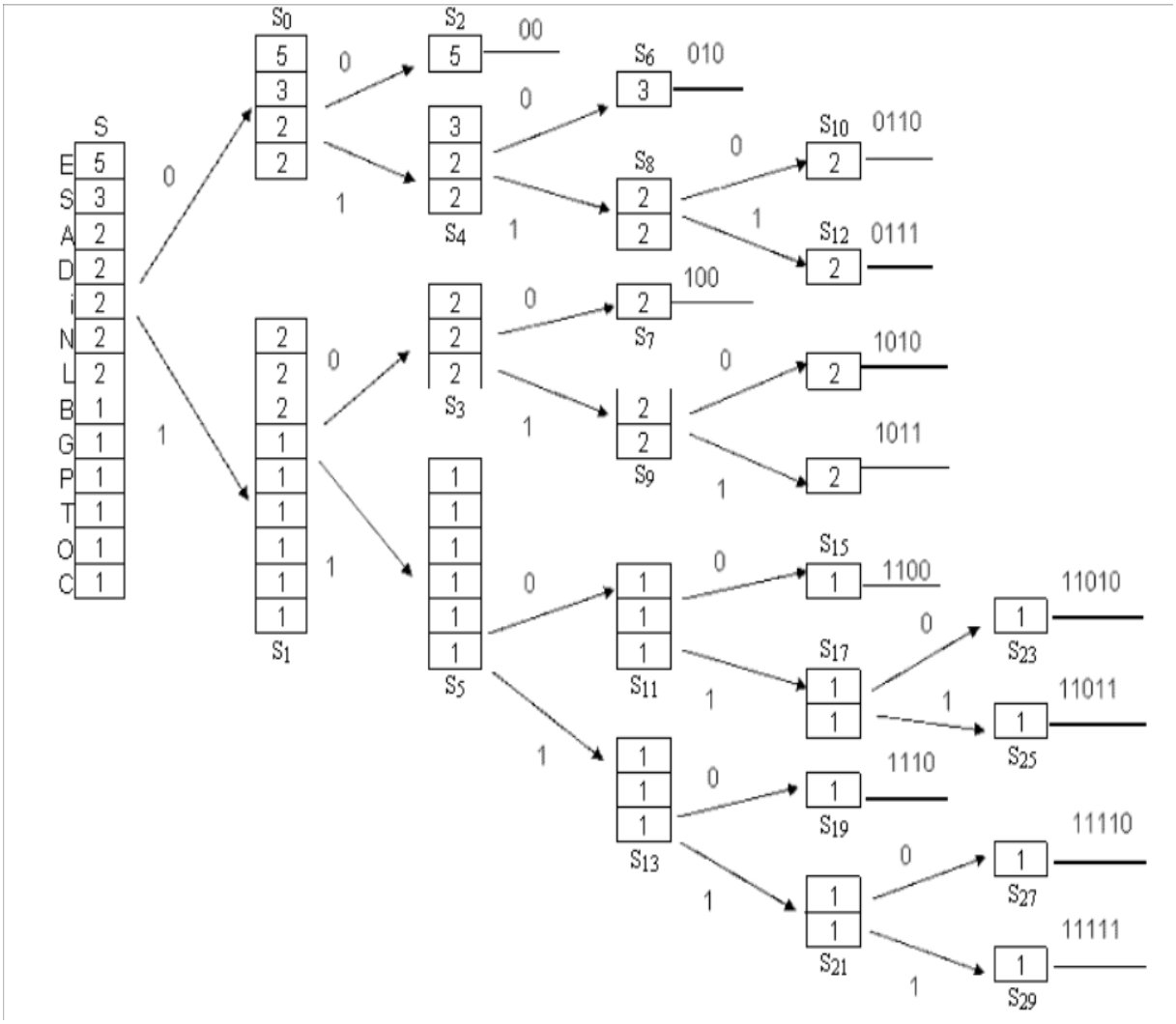


Figure 1.3 Exemple algorithme de Shannon-Fano.

V. 1.2. Par substitution [6] [11] [19]

Le codage de Huffman est efficace quand il y a un petit nombre de types de messages dont quelques-uns Couvrent une proportion importante du texte .Quand le nombre de types de messages augmente et que les fréquences sont plus uniformes, le gain est négligeable.

Souvent, le nombre de messages différents est indéterminé à l'avance (par exemple quand il s'agit de mots dans un texte simple) ou trop grand pour justifier un codage à longueur variable (par exemple, le jeu de caractères ASCII). Dans ces cas, on effectue une compression en remplaçant seulement des séquences choisies de texte par des codes plus courts.

Il y a plusieurs façons de procéder, exemples :

- parcourir le texte et remplacer les séquences redondantes par des séquences plus courtes (l'algorithme RLE).
- analyser préalablement le texte pour déterminer les groupes à remplacer et les codes qui les remplacent (l'algorithme LZW).

V. 1.2.1. L'algorithme RLE (Run Length Encoding)

RLE est un algorithme de compression de données qui est utilisé par la plupart des formats de fichiers bitmaps tels que TIFF, BMP et PCX. Il a été créé pour compresser n'importe quel type de données sans tenir compte de l'information qu'elle contient.

- **Algorithme de compression :**

- Recherche des caractères répétés plus de n fois (n fixé par l'utilisateur).
- Remplacement de l'itération de caractères par:

1. un caractère spécial identifiant une compression
2. le nombre de fois où le caractère est répété
3. le caractère répété

- **Algorithme de décompression :**

Durant la lecture du fichier compressé, lorsque le caractère spécial est reconnu, on effectue l'opération inverse de la compression tout en supprimant ce caractère spécial.

- **Exemple** : **AAAAARRRRRROLLLLBBBBBUUTTTTTT**

- ✓ On choisit comme caractère spécial : **@** et comme seuil de répétition : 3
- ✓ Après compression : **@5A@6RO@4L@5BUU@6T**
- ✓ Gain : 11 caractères soit 38%.

- **Utilité**

Essentiellement pour la compression des images (car une image est composée de répétitions de pixels, de couleur identique, codés chacun par un caractère).

- **Caractéristiques de compression :**

  - ✓ algorithme très simple
  - ✓ taux de compression relativement faible (40%).

V. 2. La compression par dictionnaire (substitution de facteurs) [6] [11] [19]

Il consiste à remplacer les séquences (les facteurs) par un code plus court qui est l'indice de ce facteur dans un dictionnaire (exemple : l'algorithme LZW).

V. **2.1 L'algorithme LZW (Lempel Ziv Welch)*:***

LZW (Lempel-Ziv-Welch) est un algorithme de compression de données sans perte. Il s'agit d'une amélioration de l'algorithme LZ78 inventé par Abraham Lempel et Jacob Ziv en 1978.

Cet algorithme de compression consiste à réaliser la construction d'un dictionnaire. Les caractères inférieurs à 256 sont initialement présents dans le dictionnaire. À mesure que l'algorithme examine le texte, il ajoute de nouvelles chaînes de caractère dans de dictionnaire.

- ✓ Ce principe forme la base de la compression LZW (Lempel-Ziv-Welch). La méthode est semblable à la méthode RLE, mais appliquées à des suites d'octets.
- ✓ Cet algorithme réduit la taille des chaînes de caractères (c'est-à-dire les mots) récurrents.

- **Algorithme de compression :**

Cet algorithme utilise un dictionnaire c'est-à-dire une table de données contenant des chaines de caractères.

Au cours du traitement de l'information, les chaînes de caractères sont placées une par une

dans le dictionnaire. Lorsqu'une chaîne est déjà présente dans le dictionnaire, son code de fréquence d'utilisation est incrémenté. Les chaînes de caractères ayant des codes de fréquence élevés sont remplacées par un " mot " ayant un nombre de caractères le plus petit possible et le code de correspondance est inscrit dans le dictionnaire.

- **Algorithme de 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.

- **Utilité et caractéristiques de compression :**

Cette méthode est peu efficace pour les images mais donne de bons résultats pour les textes et les données informatiques en général (plus de 50%).

- **Exemple**

Le tableau suivant illustre le fonctionnement de l'algorithme LZW.

- ✓ On a la chaine suivante: TOBEORNOTTOBEORTOBEORNOT
- ✓ Le résultat de l'exécution de l'algorithme de compression sur la chaîne précédente donne:

C	W	WC	Sortie	Dictionnaire
T		**T**		
O	**T**	**TO**	**T**	**TO=**
B	**O**	**OB**	**O**	**OB=**
E	**B**	**BE**	**B**	**BE=**
O	**E**	**EO**	**E**	**EO=**
R	**O**	**OR**	**O**	**OR=**
N	**R**	**RN**	**R**	**RN=**
O	**N**	**NO**	**N**	**NO=**
T	**O**	**OT**	**O**	**OT=**
T	**T**	**TT**	**T**	**TT=**
O	**T**	**TO**		

B	**TO**	**TOB**		**TOB=**
E	**B**	**BE**		
O	**BE**	**BEO**		**BEO=**
R	**O**	**OR**		
T	**OR**	**ORT**		**ORT=**
O	**T**	**TO**		
B	**TO**	**TOB**		
E	**TOB**	**TOBE**		**TOBE=**
O	**E**	**EO**		
R	**EO**	**EOR**		**EOR=**
N	**R**	**RN**		
O	**RN**	**RNO**		**RNO=**
T	**O**	**OT**		
	OT			

Tableau 1.3 exemple de l'algorithme LZW.

Après la compression, nous obtenons une séquence de codes de 9 bits sur la sortie:

TOBEORNOT.

- **Remarque**

Les programmes de compactage les plus performants combinent la puissance des algorithmes de type LZW et l'efficacité des algorithmes statistiques pour compresser les motifs redondants. La combinaison est simple :

- ✓ Aux algorithmes de type dictionnaire de détecter les redondances dans les fichiers et de constituer les dictionnaires de motifs redondants.
- ✓ Aux algorithmes statistique de trouver les codages les plus concis pour les termes contenus dans les dictionnaires.

V. 3. La compression par anti dictionnaire [6]

Un antidictionnaire est un ensemble de mots qui n'apparaissent pas dans le texte.

- **Exemple :** Soit un texte (binaire) que l'on souhaite compresser. Imaginons que le mot '1001' soit dans l'anti-dictionnaire. Dès lors, on pourra coder la séquence '1000' par '100'. En effet, sachant que le mot '1001' est dans l'anti-dictionnaire, c'est donc que ce

même mot n'est pas dans le texte à coder. Donc seul un '0' peut suivre la séquence '100' dans le texte.

V. 4. La compression symétrique/asymétrique [13]

V. 4.1 La compression symétrique

Un algorithme de compression est dit symétrique lorsque le codeur et le décodeur utilisent le même procédé. Le temps d'exécution est donc quasiment égal pour les deux étapes. Par exemple, une application de transmission de données où la compression et la décompression sont les deux faits en temps réels sera généralement implémentée avec un algorithme symétrique si l'on veut atteindre la plus grande efficacité.

V. 4.2 La compression asymétrique

Une compression asymétrique est recherchée, lorsque le décodage doit être beaucoup plus rapide que la compression, par exemple, imaginons que nous ayons une base de données où les données seront compressées une seule fois mais décompressées un grand nombre de fois pour la consultation, alors on pourra certainement tolérer un temps beaucoup plus grand pour la compression, dans le but de trouver le taux de compression le plus élevé, que pour la décompression, où là, la vitesse est prédominante. Un algorithme asymétrique qui utilise beaucoup plus de temps CPU pour la compression mais qui est beaucoup plus rapide à la décompression serait un bon choix dans ce cas là. Un exemple classique est l'encyclopédie Encarta de Microsoft.

V. 5. La compression physique/logique [21]

La distinction entre compression physique et logique est faite sur la base de comment les données sont compressées ou plus précisément comment est-ce que les données sont réarrangées dans une forme plus compacte.

V. 5.1 La compression physique

La compression physique agit directement sur les données; Cette méthode produit typiquement des résultats incompréhensibles qui apparemment n'ont aucun sens.

Le résultat d'un bloc de données compressées est plus petit que l'original car l'algorithme de compression physique a retiré la redondance qui existait entre les données elles-mêmes.

V. 5.2 La compression logique

La compression logique par contre est effectuée par un raisonnement logique en substituant une information par une information équivalente. Changer "United State of America" en "USA" est un bon exemple de substitution logique car "USA" est dérivé directement de l'information contenue dans la chaîne "United State of America" et garde la même signification.

V. 6. La compression avec/sans perte [21]

V. 6.1 La compression sans perte

Une bonne partie des schémas de compression utilisés sont appelés **sans pertes**, cela signifie que lorsque des données sont compressées et ensuite décompressées, l'information originale contenue dans les données a été préservée. Aucune donnée n'a été perdue ou oubliée. Les données n'ont pas été modifiées.

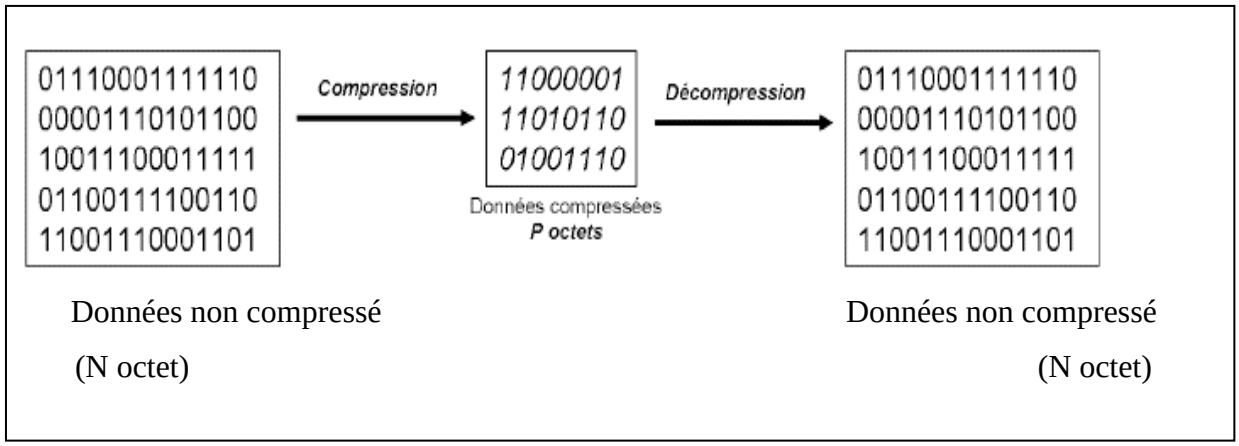


Figure 1.4 principe de la compression sans perte [20] .

V. 6.2. La compression avec perte

La compression avec perte permet d'éliminer quelques informations pour avoir le meilleur taux de compression possible, tout en gardant un résultat qui soit le plus proche possible des données originales.

Par exemple les détails d'une image, ou les fréquences inaudibles pour un fichier sonore. Certains éléments sont quasiment imperceptibles par l'humain, il est alors intéressant de profiter de cette « faiblesse » pour réduire la taille du fichier. La qualité finale du média dépendra donc de la quantité de données altérée, le but étant de trouver un bon compromis entre « Qualité » et « Taux de compression ».

V. 7. Codage adaptatif/semi adaptatif/non adaptatif [21]

Certains algorithmes de compression comme le codage de Huffmann, sont basés sur des dictionnaires spécifiques à un type de données : ce sont des encodeurs **non adaptifs**. Les occurrences de lettres dans un fichier texte par exemple dépendent de la langue dans laquelle celui-ci est écrit.

Un encodeur **adaptif** s'adapte aux données qu'il va devoir compresser, il ne part pas avec un dictionnaire déjà préparé pour un type de données mais il détermine la dépendance des données en construisant leur dictionnaire à la volée comme LZW ou Huffmann dynamique.

Enfin un encodeur **semi-adaptif** construira celui-ci en fonction des données à compresser : il construit le dictionnaire en parcourant le fichier, puis compresse ce dernier. Un dictionnaire optimal est construit avant qu'un quelconque encodage soit effectué.

VI. La compression d'image

Pour comprendre comment fonctionne la compression d'image nous devons tous d'abord définir quelque notion.

VI. 1. Définitions [7]

a- L'image

En informatique une image est composée d'un ensemble de points, appelés pixel (Picture Element).ces pixels sont regroupés en lignes et en colonne afin de former un espace a deux dimensions. Chaque point sera représenté par ses coordonnées (X, Y), avec X l'axe orienté de gauche à droite, et Y l'axe orienté de haut en bas.

b- Notion de pixel

C'est le Plus petit élément significatif d'une image numérique. Chaque pixel porte une couleur, décomposée en rouge, vert et bleu. La juxtaposition des pixels, à la manière d'une mosaïque, crée une image (qui peut représenter du texte).

Le pixel est utilisé comme unité de mesure de la taille d'affichage des écrans. Un pixel n'est pas forcément carré. Sa forme dépend du support d'affichage utilisé, de ses réglages et de la technologie.

c- Représentation de la couleur

En plus de ces coordonnées planaires, un pixel se caractérise par sa pondération, appelée aussi profondeur de codage, qui représente sa couleur ou son intensité. Cette valeur peut être codée sur un nombre n différents de bits (ou octet) selon les méthodes de codage de la couleur utilisées.les standards les plus répandus sont n=1 bit (noir ou blanc), n=4(16 couleurs ou 16 niveaux de gris), n=8bits (256 couleurs ou 256 niveaux de gris)… etc.

On appelle la palette de couleur, l'ensemble des couleurs que peut contenir une image.

VI. 2. Les Différentes formes d'images [7] [8]

Nous avons vu auparavant qu'une image était constituée d'un ensemble de points, nommées pixels, de position et de couleurs différentes. C'est ce qu'on appelle une image du type « bitmap », ou en français « tableau données binaire ».

Cependant, il existe une autre forme d'image, appelée image vectorielle. Les images vectoriels sont des représentations géométrique (cercle, rectangle, droite…), et grâce a cette propriété on peut coder une image entière en formules mathématique. Ainsi il est possible de leur appliquer des transformations géométriques.

VI. 3. Le codage de la couleur : [7] [8]

Pour représenter sur un périphérique externe, un écran par exemple, la couleur d'un pixel, nous avons besoin de codifier la couleur sur un ou plusieurs octets, nous allons donc présenter les codages de couleur les plus répandus, qui se différencient par leur mode de représentation de La couleur.

a- Le codage RGB

C'est le format le plus connu, Ce codage réserve un octet pour chaque couleurs (Rouge,Bleu et Vert).aussi, nous obtenons 256 intensités de rouge (2^8), 256 intensités de bleu, 256 intensités de vert, soient 16 777 216 possibilités théoriques de couleurs différentes.

b- **Le codage TSL (ou HSL)**

C'est un modèle de représentation dit « naturel », c'est-à-dire proche da la physiologique de la couleur de l'œil humain, ce code décompose la couleur en trois critère :

- La teinte correspond à la couleur de base.

- La saturation, décrivant la pureté de la couleur, c'est-à-dire son caractère vif ou terne.
- La luminance, indiquant la brillance de la couleur c'est-à-dire son aspect claire ou sombre.

c- **Le codage CMYK**

Il décompose les couleurs en trois couleurs (cran, magenta et jaune).ce codage réalise ensuite le même procédé que le codage RGB. La lettre K désigne le terme noir pur, cette dernière a été ajoutée pour réaliser du noir.

d- **Le codage CIE**

Utilise la chromaticité et la luminance. Puisque les couleurs peuvent êtres perçues différemment selon les individus et qu'elles peuvent êtres affichées différemment selon les périphériques d'affichage.

e- **Le codage YUV**

Ce codage est utilisé dans les standards PAL et SECAM, il a été conçu pour être reconnus par les télévisions en noir et blanc, qui convertissaient les couleurs en un dégradé de gris.

VI. 4. Codage et compression

Le premier principe de la compression d'mage consiste à éliminer les informations redondantes de l'image, c'est-à-dire à coder de manière plus simple les informations les plus répétées. Ainsi d'éliminer uniquement les éléments les moins visibles de l'image.

Dans le but de la compression d'images plusieurs recherches ont essentiellement porté sur des algorithmes de compression et ont donné naissance à des normes qui permettaient des économies de l'ordre de 10 à 90%, mais qui ont été très vite insuffisantes devant les problèmes que posaient le stockage de milliers d'images, ce qui a rendu nécessaire la mise en place sur le plan international de groupes de coordination et d'étude, chargés de mettre au point des standards adaptés à ces applications afin de rendre cohérents et compatibles les échanges d'informations sur les canaux de communication.

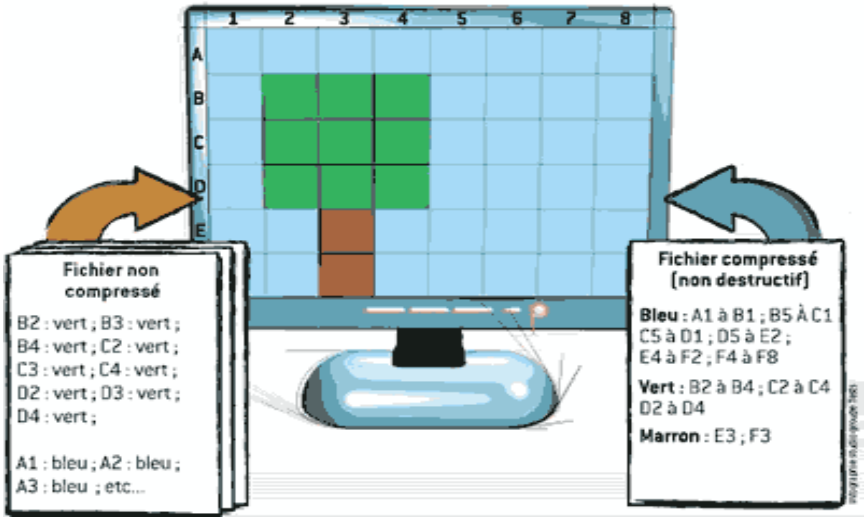


Figure 1.5 principe de la compression d'image (l'élimination des redondances)[35].

VI. 4.1. La compression JPEG (Joint Photographic Expert Group) [9]

La norme JPEG (Joint Photographic Experts Group) est conçue par le groupe ISO (International Standards Organisation) et le groupe CEI (Commission Electronic International). Elle est destinée à la compression des images fixes en couleurs et à niveaux de gris en vue de leurs stockages sur les supports numériques.

Elle a été réalisée dans la perspective de couvrir les applications les plus diversifiées en tenant compte des contraintes réalistes par rapport aux applications les plus visibles : publication, transmission, banques d'images.

La compression JPEG consiste à effectuer une dégradation de l'image indiscernable à l'œil, de façon à offrir un taux de compression beaucoup plus intéressant que les autres méthodes, donc à permettre de réduire considérablement l'espace occupé par le fichier sur le disque.

Pour ce faire, l'image est décomposée en blocs qui vont ensuite subir diverses opérations pour diminuer la taille de l'image initiale. Cette compression est donc une compression avec perte car elle subit une perte définitive d'information, même si il est possible de revenir à une image proche de l'image initiale avec certains procédés.

Voici le procédé de compression :

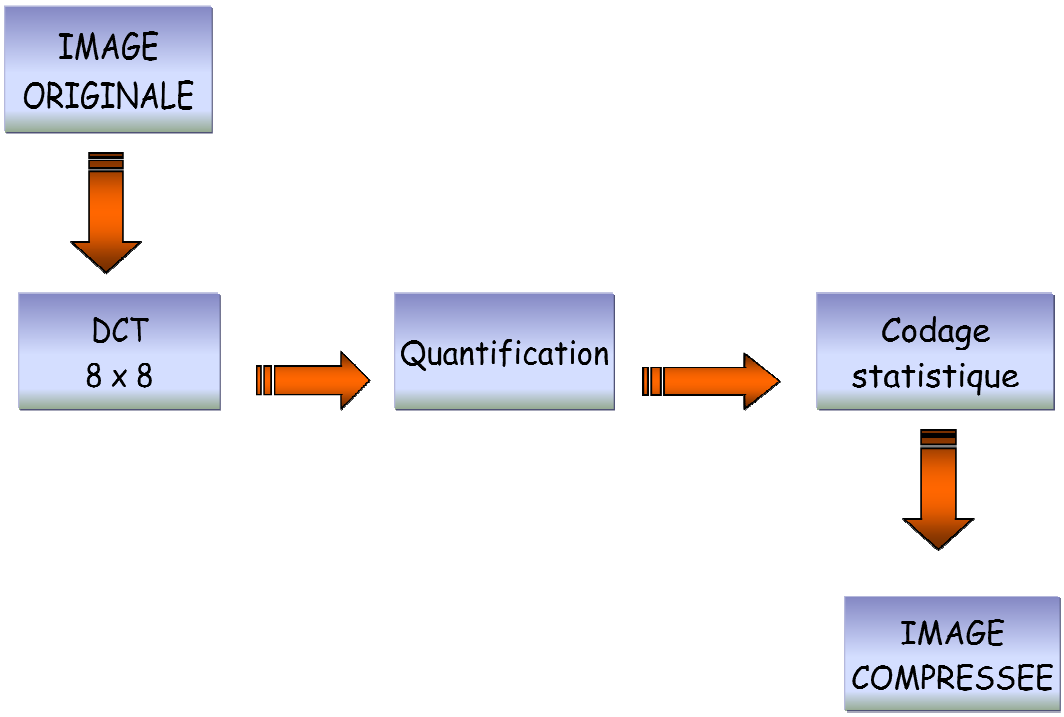


Figure 1.6 la compression JPEG [9].

- **Découpage en blocs de 8×8 pixels et transformation DCT :**

Une image informatique est constituée de points de couleur uniforme : ce sont les pixels. Ils sont caractérisés par leur position dans l'image, et par leur couleur.

On découpe ensuite cette matrice en sous matrice de 8×8 pixels.

A chacune de ces matrices on a applique ensuite la transformation DCT (Discrete Cosine Transform, ou Transformée en Cosinus Discrète). Cette technique est similaire de à la transformée de Fourier : transformation en somme de sinus et de cosinus de différentes fréquences et amplitude, mais pour la DCT on ne prend en compte que les cosinus.

- ✓ La transformée DCT s'exprime mathématiquement par :

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

Équation 1 : Transformée DCT directe.

✓ La constante C vaut :

$$C(x) = \begin{cases} \frac{1}{\sqrt{2}} & \text{pour } x = 0 \\ 1 & \text{pour } x > 0 \end{cases}$$

Equation 2 : Définition de la constante C

➢ **La Quantification**

La quantification est l'étape dans laquelle on perd réellement des informations (et donc de la qualité visuelle), mais c'est celle qui fait gagner beaucoup de place (contrairement à la DCT, qui ne compresse pas).

La DCT a retourné, pour chaque bloc, une matrice de 8×8 nombres (dans l'hypothèse que les blocs de l'image font 8×8 pixels). La quantification consiste à diviser cette matrice par une autre, appelée matrice de quantification, et qui contient 8×8 coefficients savamment choisis par le codeur.

Le but est ici d'atténuer les hautes fréquences, c'est-à-dire celles auxquelles l'œil humain est très peu sensible. Ces fréquences ont des amplitudes faibles, et elles sont encore plus atténuées par la quantification (les coefficients sont même ramenés à 0).

✓ Voici le calcul permettant la quantification :

$$F^*(u,v) = \left\lfloor \frac{F(u,v) + \left\lfloor \frac{Q(u,v)}{2} \right\rfloor}{Q(u,v)} \right\rfloor$$

$Q(i, j) = 1 + (1 + i + j) \times Fq.$ avec Fq facteur de qualité et Q le pas de quantification.

Équation 3 : Calcul de la quantification.

- **Codage statistique**

On peut ici utiliser un code statistique tel que le codage de Huffman.

VI. 4.2. La compression Fractale[7][9]

Les fractales sont des courbes mathématiques infiniment complexe.si le mot fractale lui-même ne date que depuis 1981, Dans les années 1950, un mathématicien nommé Benoit Mandelbort, découvrait ce qu'on appelle aujourd'hui, la fractale.

Une image fractale est une répétition d'un motif plus au moins rétréci, plus au moins transformé, Barnsley eut l'idée de rechercher dans n'importe quelle image, des formes similaire. Ainsi au lieu de coder toute l'image, la compression fractale n'encode que les motifs, ainsi que les transformations a réaliser.

- ✓ **Principe :**

Les images fractales sont générées à partir d'une équation et de quelques paramètres. La compressé d'une telle image est basé sur deux grandes idées, la conception fractale de l'image et le théorème du point fixe, la première de ces idées est d'exploiter les similarités que l'on peut constater entres différentes parties d'une même image. On postule que chaque morceau d'image peut s'exprimer a l'aide d'une transformation affine contractante d'un autre morceau de la même image.

VI. 4.3. La Compression par Ondelettes [7] [9]

La théorie des ondelettes a été inventé par le mathématicien hongrois Alfred Haar dans les années 1910.une ondelette est une transformation de fonction qui oscille principalement dans un intervalle restreint.

La transformation par Ondelettes est une technique qui consiste à décomposer une image en une multitude de sous-bandes, c'est à dire des images de résolution inférieure. Nous distinguons 4 étapes différentes pour procéder à la transformation:

- ✓ **principe :**

Une image est formée de carrés de 256 pixels par 256 et chaque pixel est constitué de couleurs différentes. On peut donc modéliser cette image en une matrice carrée de 256 lignes et 256 colonnes où chaque coefficient représente le code d'une couleur.

On multiplie maintenant cette matrice par une matrice creuse (contenant beaucoup de zéros), du type :

A= $\begin{bmatrix} 1/2 & 0 & 1/2 & 0 \\ 1/2 & 0 & -1/2 & 0 \\ 0 & 1/2 & 0 & 1/2 \\ 0 & 1/2 & 0 & -1/2 \end{bmatrix}$

Exemple :

Soit M= $\begin{bmatrix} a & b & c & d \\ e & f & g & h \\ i & j & k & l \\ m & n & o & p \end{bmatrix}$ Þ P=M´A=$\begin{bmatrix} (a+b)/2 & (c+d)/2 & (a-b)/2 & (c-d)/2 \\ (e+f)/2 & \dots & (e-f)/2 & \dots \\ \dots & \dots & \dots & \dots \\ \dots & \dots & \dots & \dots \end{bmatrix}$

Dans la première moitié de la matrice on a l'échantillon principal, et dans la deuxième on a les détails. On veut maintenant effectuer cette opération sur les colonnes, pour ce fait on calcule N = ${}^{t}A \acute{} M \acute{} A$. On applique maintenant le coefficient de précision à la matrice, c'est à

dire que l'on supprime tous les coefficients inférieurs à e. On obtient maintenant une nouvelle matrice N ' et pour retrouver la matrice originale on calcule M' = ${}^tA^{-1}$ ´ N' ´ A^{-1}.

Mais ces calculs sont longs et on peut les réduire considérablement en remarquant que ´A est une matrice rectangulaire ($tA = A^{-1}$).

Pour augmenter le taux de compression, on utilise plusieurs fois l'algorithme, mais on remarque que c'est toujours le quart de la matrice en haut à gauche qui comporte des coefficients nécessitant d'être réduits. L'algorithme est donc appliqué à cette partie.

VII. La compression vidéo

La compression vidéo est une méthode de compression de données, qui consiste à réduire la quantité de données, en minimisant l'impact sur la qualité visuelle de la vidéo. L'intérêt de la compression vidéo est de réduire les coûts de stockage et de transmission des fichiers vidéo.

VII. 1. La notion de vidéo [10]

En latin vidéo « je vois» technique de transformation d'images animées en signaux électronique (signaux vidéo), destinée à permettre leur diffusion ou leur enregistrement.

- ✓ L'œil humain a comme caractéristique d'être capable de distinguer environ 20 images par seconde. Ainsi, en affichant plus de 20 images par seconde, il est possible de tromper l'œil et de lui faire croire à une image animée.

VII. 2. Principe de compression [14]

Le principe de la compression vidéo repose sur la forte redondance d'informations dans une image et d'une image sur l'autre. Il en résulte deux types de compression : une compression spatiale et une compression temporelle.

VII. 2.1. La compression spatiale

Dans ce type de compression chaque image est prise indépendamment des autres. On peut diminuer la redondance présenté dans les images (le fait que des pixels voisins soient corrélés), en codant chaque image séparément en JPEG (*Joint Photographic Experts Group*). Cette approche est parfois utilisée lorsqu'on a besoin de pouvoir accéder de façon aléatoire à chaque image individuellement, comme par exemple lors d'un montage vidéo.

VII. 2.2. La compression temporelle :

Dans une séquence vidéo on peut constater que deux images qui se suivent sont quasiment identiques. Le but est alors de ne stocker que ce qui est modifié lors du passage d'une image à une autre.

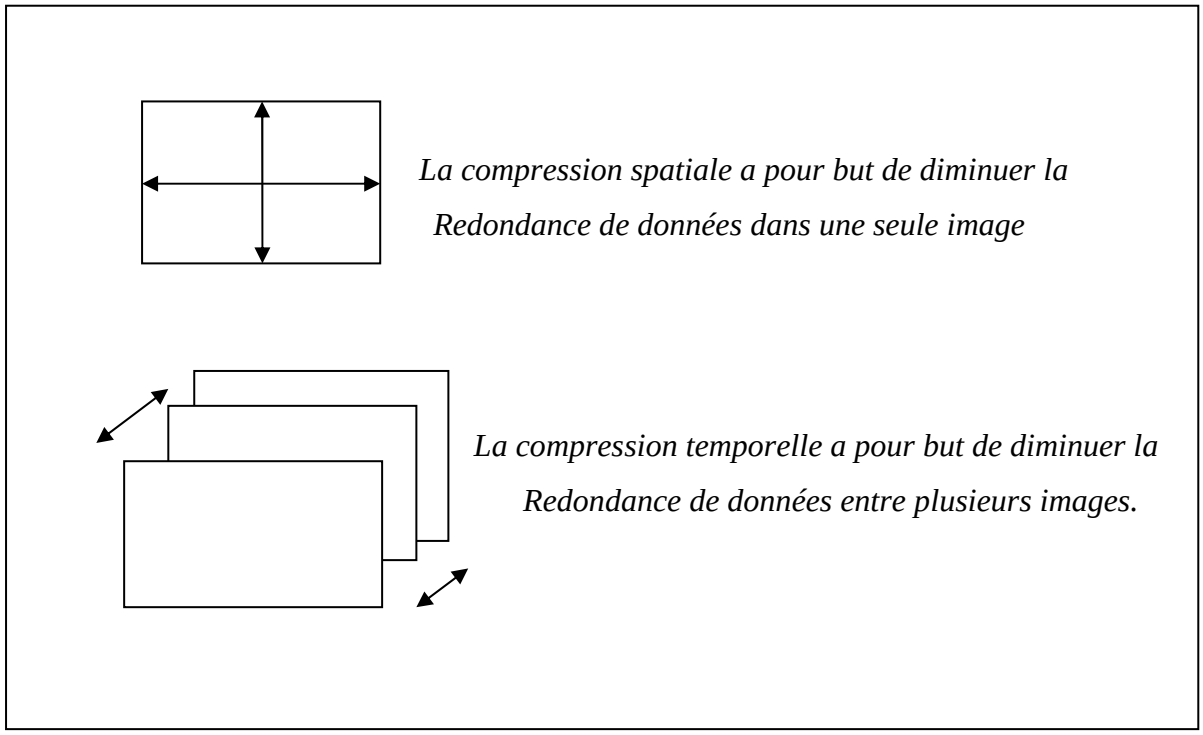


Figure 1.7 compression spatiale et compression temporelle [14].

***VII. 3. La compression MPEG (Moving Picture Experts Group)* [14][22][6]**

La norme MPEG (Moving Picture Expert Group) définit une compression pour une succession d'images dans le temps, accompagnée d'une bande sonore. Elle a été créée pour les applications multimédias sur PC, il est actuellement utilisé par beaucoup d'appareils multimédias, comme les appareils photos digitaux.

VII. 3.1. Les différentes normes :

- **Le MPEG – 1** :

La norme MPEG-1 est un standard pour la compression des données vidéo et des pistes audio qui y sont associées. Elle est finalisée en 1992, Le but des chercheurs qui ont travaillé sur cette première norme était de stocker et de reproduire de la vidéo avec le son associé sur des supports de stockage, en qualité magnétoscope (320 x 240), avec un débit maximum égal à 1,5 Mbits/s. Concrètement, MPEG-1 est devenu par la suite la norme de stockage de vidéos sur CD-ROM au format CD-I ou CD-vidéo.

- **Le MPEG – 2** :

La norme MPEG-2 est un standard principalement dédié à la télévision numérique. elle a été finalisée en 1996, Le MPEG-2 intégra la compression de la télévision à haute définition (TVHD). Aujourd'hui, MPEG-2 est aussi le format utilisé pour stocker les films sur DVD.

- **Le MPEG – 4** :

Est un format conteneur qui permet de coder des objets audio et vidéo, ainsi que du contenu 3D. Les objectifs de cette norme sont assez nombreux et touchent de vastes domaines. En fait, MPEG-4 a pour ambition de fusionner trois mondes : l'informatique, les télécommunications et la télévision. Cette norme s'appuie sur les succès futurs de la télévision numérique, des applications graphiques interactives et du multimédia (web), son but étant d'assurer une standardisation technologique à tous les niveaux : production, distribution et diffusion. Ses domaines d'application sont immenses : communication temps réel (visiophone), multimédia mobile, téléconférence, post-production (cinéma et télévision), stockage (DVD) etc.…

- **Le MPEG – 7** :

Le MPEG-7 est en élaboration depuis 1997, il a pour principal objectif de simplifier et d'optimiser la recherche de fichiers multimédia. Pour cela, le MPEG-7 va spécifier une description standard de différents types d'informations multimédia. Cette description devra être associée au contenu lui-même pour permettre la recherche rapide et efficace des informations qui intéressent l'utilisateur. Cela peut s'appliquer aussi bien à la vidéo qu'à du son, des images, des graphiques, des animations 3D, ou encore à la façon dont des objets vidéo sont combinés dans une séquence.

- **Le MPEG – 21** :

Le MPEG – 21 est en élaboration depuis 2000, elle ajoute notamment à MPEG-7 une couche "sécurité" en proposant tout un éventail de solutions permettant des échanges de matériaux audiovisuels en toute sécurité (déclaration, identification, protection). MPEG-21 redéfinit également la structure interne des séquences audiovisuelles en optimisant les interactions possibles entre les objets audiovisuels, et entre les utilisateurs et ces mêmes objets.

- **Le H.263 (codec vidéo) :**

Le H.263 été développé pour transmettre des vidéos sur des réseaux à très bas débit. Il a été ensuite adapté pour les visioconférences. Le H.263 est utilisé dans les videoconférences et pour les vidéos sur les téléphones portables de 3ème génération.

- **Le H.264 (codec vidéo)**

Le H.264 connu sous le nom de AVC (Advanced Video Coding) est une norme de codage vidéo qui compresse beaucoup plus efficacement les vidéos que les normes précédentes (H.263) et fournit plus de flexibilité aux applications. Il a été développé pour être utilisé avec du MPEG. Ce codec est utilisable sur beaucoup de réseaux et de systèmes différents (télévision, téléphonie, etc…). Il est possible que dans le futur, le H.264 soit utilisé pour la location de films à travers internet.

VIII. La Compression des donnés audio

VIII. 1. La notion de son [15] [23]

Le son est une somme de vibrations, produite par des codes vocaux, un haut-parleur, etc...., ces vibrations ont une fréquence, mesurée en Hertz.

- ✓ L’oreille humaine est un récepteur ne percevant que certaines fréquences : la bande 20Hz – 20 KHz.
- ✓ Les vibrations sont codées dans un fichier par une suite de 0 et de 1. pour l’encoder on mesure la hauteur de la courbe sonore des milliers de fois par seconde. C’est l’échantillonnage.

VIII. 2. Principe de compression

La compression de son est une forme de compression de données. Celle-ci a pour but de réduire la taille d'un fichier audio afin de répondre au besoin de bande passante de la transmission des flux audio numériques et de la taille de stockage de fichiers audio.

- ✓ La compression audio supprime les hautes fréquences quasiment inaudibles, elle diminue ainsi la fréquence d'échantillonnage.
- ✓ On distingue la compression audio sans perte (privilégier dans l'archivage des données audio, ex :WAV, AIFF, FLAC, CDA) et la compression audio avec perte (facilite le stockage sur des supports, ex : MP3, MP4, WMA) .

VIII. 3. Les formats de compression audio : [15]

- ➢ **Le WAV**

Le format WAV joue le rôle de conteneur, il n'est donc pas un véritable format audio.il permet de recevoir divers flux de compression ou de codec comme le MP3, le WMA ou le PCM, c'est ce derniers que l'on retrouve le plus souvent utilisé . Ce format a été développé par Microsoft et IBM .limité dans sa taille et relativement ancien, le WAV est le format audio le plus utilisé sur la plateforme Windows.

- ➢ **L'AIFF (Audio Interchange Format File)**

Idem que le son WAV mais développé par la société Apple.la conversion du fichier peut se faire depuis un logiciel. L'avantage est qu'il ne subit aucune perte qualitative .ce format représente la particularité d'offrir un débit binaire constant de 1411 Kbits (Kbits/s) pour la stéréo et de 705 Kbits/s pour une source en mono.

- ➢ **Le FLAC (Free Lossless Audio Codec)**

Le FLAC est un codec de compression audio qui présente l'avantage d'offrir une compression non-destructive.il utilise une méthode de compression qui diminue le débit sonore ainsi que la capacité de stockage sans retirer de données du flux audio originel.

Le taux de compression dépend des informations à traiter mais s'échelonne entre 30 et 60% de réduction.

- **Le CDA (Compact Disc Audio)**

C'est le format des contenus audio se trouvant sur un CD audio. C'est un format naturellement nom compressé de qualité CD, présentant le désavantage d'être lourd. Ce format est le plus souvent compressé afin de le stocker dans un baladeur numérique par exemple.

- **Le MP3 (MPEG-1 Layer 3)**

Le plus connus et le plus utilisé des formats de compression audio. Ce format est capable de compresser jusqu'à 12 fois le fichier original mais le son devient fortement inaudible a mesurer que le débit binaire diminue, pour une écoute acceptable en MP3, on s'autorise un débit allant de 128Kbits par secondes a 192 Kbits/s.

- **Le WMA (Windows Media Audio)**

Le format WMA est un format propriétaire développé par Microsoft. On retrouve dans ce format plusieurs formes de compression allant du « standard » au « sans perte » et offre des débits allant jusqu'à 320 Kbits/s.

IX. Les critères de choix d'un algorithme de compression [9]

Certains algorithmes sont très performants sur un type de données, mais complètement inutiles sur d'autres. Par exemple, un algorithme combinant le codage « par plages » et « à longueur variable » seront très performants sur des fichiers textes, le codage MP3 sera efficace sur les fichiers audio tandis que le JPEG sera utilisé pour les images et les vidéos. Il sera vu ultérieurement que plusieurs de ces algorithmes sont utilisés dans d'autres plus complexes (par exemple RLC et Huffman dans JPEG).

- ✓ **Les différents algorithmes de compression sont choisis en fonction de :**

- Leur taux de compression (rapport de la taille du fichier compressé sur la taille du fichier initial).
- La qualité de compression (sans/avec perte et alors pourcentage de pertes).
- La vitesse de compression et de décompression.

X. Exemples d'utilisation de la compression [9]

- **Selon le stockage : la compression des données est utilisées dans :**

- Les Bases de données (les fichiers du FBI contiennent environ 200 millions d'empreintes digitales).
- Les Mini-disc (les données audio y sont compressées).

- **Selon le transport :**

- Réseaux par câbles (Minitel, Internet dont la bande passante, i.e. le débit est très faible).
- Réseaux sans fil (communication par satellite avec par exemple la télévision par satellite, le téléphone portable …).

XI. Conclusion :

Les méthodes de compression ne cessent pas de se développer et de s'améliorer ou plusieurs recherches sont en cours afin d'optimiser le stockage et la diffusion à travers les différents réseaux.

Dans ce chapitre nous avons définis la compression des données ainsi que ses différents formats selon plusieurs critères, ensuite nous avons vus les différents algorithmes de compression selon les différents types des données (texte, image, vidéo, audio).

CHAPITRE 1 :

GÉNÉRALITÉS SUR LA COMPRESSION DE DONNÉES

CHAPITRE 2 :

LES MÉTHODES VLC

CHAPITRE 3 :

ANALYSE ET CONCEPTION

Chapitre 4 :

Implémentation et évaluation

I. Introduction :

Les codes a longueur variable ont été considères dès les premiers travaux de Shannon sur la théorie des communications, et sa technique de codage est la plus utilisé dans la compression de données, il s'effectue selon un algorithme bien précis, se base sur le remplacement des codes les plus court avec les symboles plus fréquents et vice versa. C'est contrairement aux méthodes à longueur constante de codage, auxquelles la compression de données est seulement possible à de grands blocs de données.

Quelques exemples des stratégies de longueur variable bien connues de codage sont Codage de Huffman, Codage de Lempel-Ziv et codage arithmétique.

II. Définitions :

II. 1. Source discrète [25]:

C'est des sources qui produisent un nombre fini d'élément appelaient symboles de source et quand il s'agit de symbole d'écriture on parle de caractère .voici certains notations des sources discrète :

X : une variable aléatoire discrète a valeurs dans A.

A : un alphabet (en pratique ensemble fini).

$A = \{x_1, x_2, \dots, x_n\}$

P(X), x ϵ A : loi de probabilité de X

p(x) = P{X=x} : la probabilité de l'évènement x

- **Source discrète à mémoire [25]:**

On prend par exemple un texte français, tel que la probabilité d'avoir la lettre « u » est très grande si la lettre précédente est « q », donc il s'agit d'une probabilité conditionnelle notée P (u/q).

- **Source discrète sans mémoire [25] :**

Une source discrète et dite sans mémoire si sa loi de probabilité ne varie pas au cours du temps c'est-à-dire la probabilité d'émission d'un caractère est indépendant de ce qui à été émis avant ou sera émise après.

II. 2. Le code source

Un code source C d'une variable aléatoire X d'alphabet A est une application :

$$C : A \rightarrow \{0,1\}^*$$
$$X \rightarrow C(x).$$

C(x) est appelé mot de code de l'élément x.

Exemple : Code1 (a1) = 1 et Code1 (a4) = 00

II. 3. La longueur moyenne du code

La longueur moyenne du code C est donnée par la relation suivante : $\boldsymbol{L(C) = \sum p(x)l(x)}$ *avec* $l(x)$ *longueur du mot de code* $C(x)$, $x \in A$.

II. 4. L'efficacité de code

Un codage est dit d'autant plus efficace que le nombre de codes possibles inutilisés est faible, est aussi dépend de la quantité d'information moyenne de la source $E(C)=\frac{H(X)}{L(C)}$

II. 5. L'extension du code

L'extension du code C est l'application qui associe a toute séquence d'éléments de A la séquence obtenu par concaténation des mots de codes associes : $C(x_1x_2 \dots x_n) = C(x_1)\, C(x_2) \dots C(x_n)$.

II. 6. Régularité

Un code est dit régulier si tous les mots de code C(x) sont distincts. Tous les codes doivent au moins être réguliers pour permettre un décodage univoque.

II. 7. Déchiffrabilité

Un code régulier est déchiffrable si pour toute suite $C(x_1)$,$C(x_2)$, …..$C(x_n)$ de mots de code il est possible de distinguer les C(x) sans ambiguïté et reconstruire ainsi les symboles Si correspondants. Pour les codes de longueur variable, cette propriété est satisfaite pour les codes dits sans préfixes, obtenus en évitant qu'un mot du code ne soit identique au début d'un autre.

II. 8. Un code à longueur fixe [27]:

Une manière simple de coder en binaire l'alphabet d'une source est d'attribuer à chaque symbole R bits. Il y a donc 2^R codes possibles et bien sûr nous avons la condition $2^R \geq K$ l'égalité étant possible lorsque le nombre K de symboles de la source est une puissance de 2. Dans le cas contraire nous aurons : $2^R-1 < K < 2^R$.

$$R = Int [\log2(K)] + 1$$

- **Nous avons :**

$$H (X) \leq \log2(K)$$

$$\rightarrow R \geq H(X)$$

$$R \geq \log2 (K)$$

L'égalité a lieu lorsque tous les symboles de la source sont équiprobables et lorsque K est une puissance de 2.

- ✓ Un codage est dit d'autant plus efficace quand le nombre de codes possibles inutilisés est faible.
- ✓ L'efficacité dépend aussi de la quantité d'information moyenne de la source. L'efficacité η d'un codage sera ainsi définie par :

$$\eta = \frac{H(x)}{R}$$ elle est exprimée en %.

III. Le Codage à longueur variable (codage de source ou codage entropique)[24] :

III. 1. Définition

Un codage entropique, également dénommé codage VLC (variable length code) car la longueur du code résultant est variable .Le codage entropique consiste, habituellement, à créer un code préfix et à l'assigner à chaque nouveau symbole (octet, par exemple) qui se produit à l'entrée.

Ce code est de longueur variable, les codes les plus courts étant attribués aux symboles les plus fréquents (contrairement au code ASCII qui a une longueur fixe).

III. 2. Les cas d'utilisations de codage source :

- Quand on manipule du texte, les caractères utilisés n'ont pas la même probabilité d'apparition. De plus il y'a une structure interne forte (la grammaire, orthographe …).

- Quand on manipule le son ou la musique, la distribution d'apparition des sons n'est pas uniforme
- Quand on manipule des images, elles possèdent également des régularités. Elles ne sont pas aléatoires.

C'est cette caractéristique des sources d'information naturelles qui incite à compresser les données.

III. 3. Les propriétées des codes à longueur variables [28] :

Des codes de longueur variable peuvent être strictement nichés par ordre de généralité décroissante comme non singuliers, uniquement décodable et instantané (préfixe librement). Les codes instantanés sont toujours uniquement décodables, qui sont à leurs tours toujours non singuliers .

III. 3.1. Code non singulier :

Un code C d'une variable aléatoire X est dit non singulier si : $Xi \neq Xj \Rightarrow C(Xi) \neq C(Xj)$ et aussi La non singularité assure le décodage d'un seul élément à la fois et assure l'intégrité de l'information non affectée par le codage.

III. 3.2. Codes uniquement décodable :

Les codes à longueur variable sont utiles pour la compression de donnée, cependant un code à longueur variable peut être non utile si les mots du code ne peuvent pas être identifiés d'une manière unique à partir de message encodé.

- **Exemple1**

On considère le code à longueur variable (0, 10, 010,101) pour l'alphabet (A, B, C, D). Un segment de message encodé comme '0100101010' peut être décodé de plusieurs manière .par exemple '0100101010' peut être interprété au minimum de manière,'0 10 010 101 0' pour donner ABCDA ou '010 0 101 010' pour donner CADC.

- ✓ **Définition**

Un code est dit uniquement décodable s'il y'a un seul chemin possible pour décoder tous messages encodé par ce code.

Le code (0, 10, 010,101) de l'exemple précédent n'est pas uniquement décodable, cependant il ne peut pas être utilisé pour la compression des données.

Le code idéal dans cette situation est un code qui n'est pas uniquement un code à longueur variable, mais il possède aussi une propriété d'auto-ponctuation.par exemple, le code à longueur variable (0, 10, 110,111) a la propriété d'auto-ponctuation. Et on remarque que les longueurs des mots de code sont les mêmes que le code (0, 10, 010,101).

- ✓ **Définition**

La propriété d'auto-ponctuation peut être vue plus clairement si on associe aux mots du code avec des nœuds d'un arbre binaire de la figure 3.1. Chaque branche gauche est considérée comme 0, et une branche droite est considérée comme 1 dans l'arbre binaire.
Durant le décodage, chaque mot de code peut être obtenu par la collection de tous les 0s et 1s de la racine à chaque feuille . chaque fois qu'une feuille est atteinte .on sait que c'est la fin d'un mot de code

III. 3.3. Code instantané :

Un code est dit instantané si aucun mot code n'est le préfixe d'un autre mot code.

quant des mots de code sont de longueur différente, c'est possible que le plus court est identique aux premiers bits du mot plus long , dans ce cas le court est dit qu'il est un préfixe du mot de code plus long.

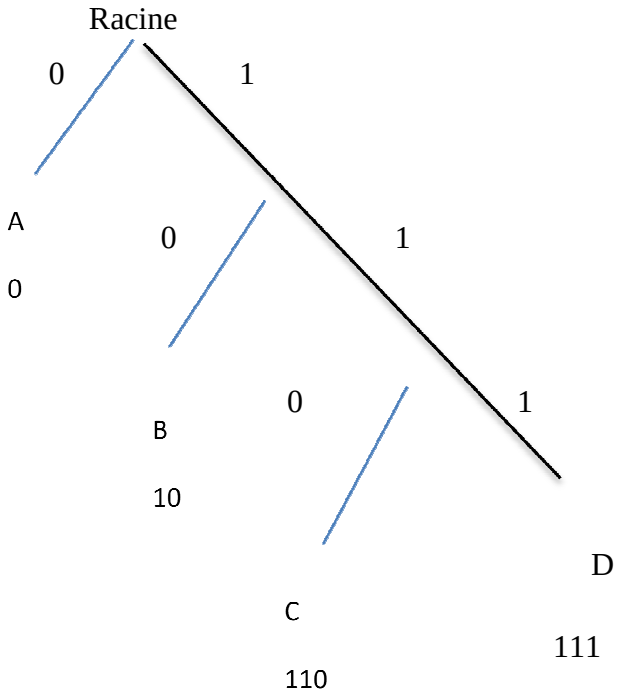


Figure 2.1:le code préfixe

- **Exemple 1**

Considérons les mots des deux codes d'un code de longueur différents : C1=010 et C2=01011.

Le plus court mot C1 est préfixe pour le mot le plus long C2 peut être obtenu on ajoutant 2 bit 11 au mot C1

- **Exemple2**

Pour montrer le code préfixe nous allons utiliser une table qui continents les trois codages possible pour une source de 4 symboles

Caractère	Probabilité	Code1	Code 2	Code 3
I	1/2	1	0	0
B	1/4	00	10	01
F	1/8	01	110	011
O	1/8	10	111	111

Tableau 2.1 exemple de code préfixe.

Supposons que nous cherchions à transmettre le message BOF

- avec le code 1, le message envoyé est : 001001. C'est ce que voit le récepteur comment peut il interpréter ?de manière correcte bien sur mais aussi 00 1 00 1 c'est-à-dire BIBI. Problème le message n'est pas décodable de manière unique. Ceci est du au fait que le 1. Code attribue à « I » est le début d'un autre code 10 attribué au O .pour éviter cette situation, il ne faut pas qu'un code soit le « préfixe » d'un autre code. Les codes qui remplissent cette condition sont des codes préfix.
- avec le code 3, le message envoyé est 01111011. Au décodage nous pouvons voir 0111…c'est-à-dire IO … mais ici nous nous rendons compte du fait que ce qui suit c'est-à-dire soit 1, soit 10, soit 101 ne sont pas des codes et donc nous pouvons revenir en arrière pour modifier l'interprétions soit 01 111 011 et retrouver le bon message .le code n'est décodable de manière instantanée.

Ceci est aussi dû au fait que le code utilisé n'est pas un code préfixe.

- le code 2 est lui un code préfixe et nous avons les deux propriétés souhaité : décodable de manière unique et de manière instantanée.

IV. L'inégalité de kraft[30] :

Les codes préfixes ont une autre propriété intéressante .pour n'importe code qui n'est pas un code préfixe, avec longueur des mots qui satisfait certains condition, on peut toujours trouver un code préfixe avec les mêmes longueurs des mots, dans l'exemple suivent qui va montrer le théorème de Kraft :

- **Exemple1**

Considérons un code binaire C= $\{c^1 \ldots\ldots c^{Lx}\}$ représentant sans erreur une source. Une condition nécessaire et suffisante pour que ce code vérifie la condition du préfixe est que

$\sum_{i=1}^{L_x} 2^{-l(ci)} \leq 1.$ (*) , Où $L(C^i)$ est la longueur du mot de code C^i

Donnons simplement le principe de la démonstration de la condition nécessaire lorsque L_x est une puissance de 2. Considérons l'arbre complet représentant l'ensemble des L_x mots d'un code C' dont tous les mots du code auraient la même longueur L_{max} telle que $Lx=2^{i\ max}$.

Ce code vérifie la relation (*) précédemment trouvée puisque .

$$\sum_{i=1}^{Lx} 2^{-imax} = 2^{imax} 2^{-imax} = 1$$

On passe de l'arbre associé au code $\mathbf{C^i}$ à l'arbre associé au code **C** en élaguant un certain nombre de branches et en créant de nouvelles. Elaguer des branches ne modifie pas le premier membre de la relation (*) puisque l'on sera toujours amené à remplacer deux termes de la forme 2^{-i} par 2^{-i+1}. Il en sera de même si l'on crée de nouvelles branches puisque l'on remplacera 2^{-i} par $2 \times 2^{-i+1}$.

- **Exemple 2**

on considère le code préfixe (0,10,110,1111) les longueurs des mots dans ce code sont 1,2,3 et 4.cependant la longueur de dernier mot de ce code peut être réduite de 4 à 3 comme (0,10,110,111) qui est aussi un code préfixe avec longueurs de ces mots 1,2,3,3,.la figure 5.1 (a) et (b) montre les arbres binaires qui sont associés au code (0,10,110,1111) et (0,10,110,111) respectivement .comme on peut voir ,si Lj est la longueur de $j^{iéme}$ mots ,avec j=1,…4,alors le niveau de la feuille de ce mot est Lj+1.

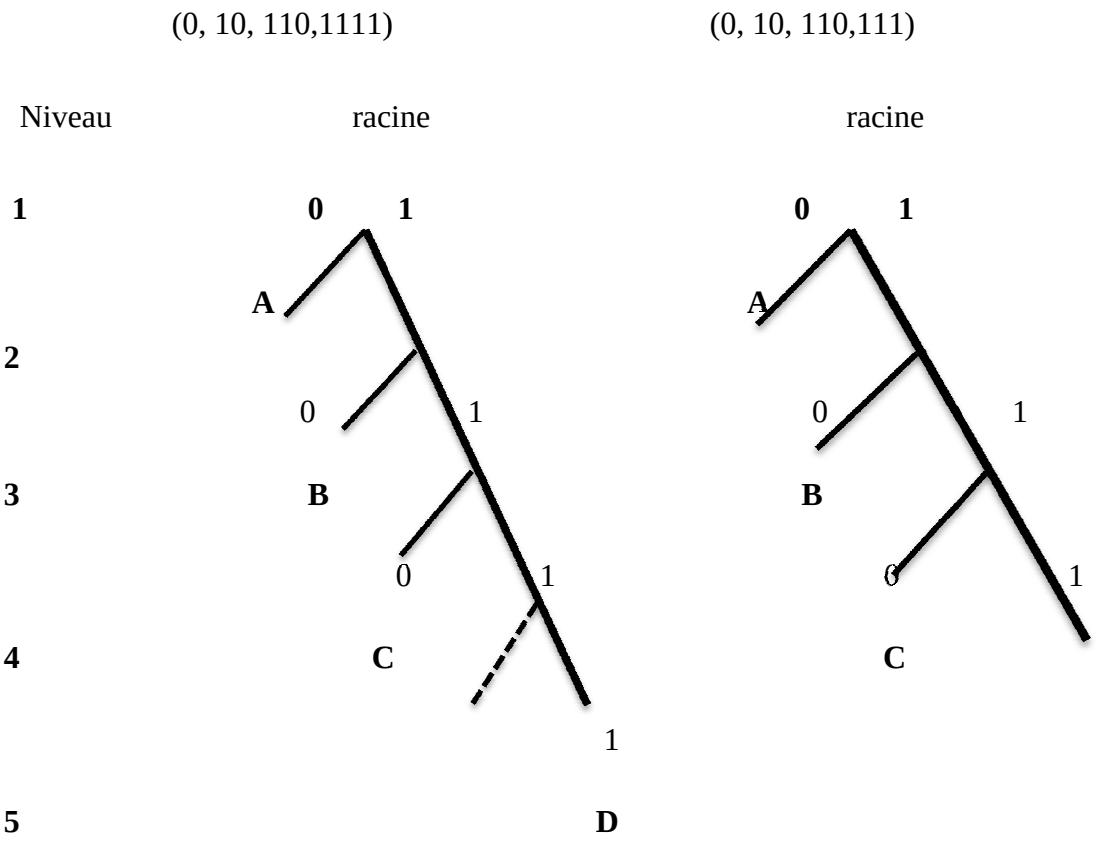


Figure 2.2 : exemple de construction de deux codes préfixes

- **Remarque**

L'inégalité de Kraft pose le nécessaire pour les longueurs des mots d'un code préfixe.si les longueurs ne satisfont pas l'inégalité de Kraft on sait qu'il n'y a pas de chance de trouver un code préfixe avec ces longueurs.

L'inégalité de Kraft ne nous dit pas comment construire un code préfixe .mais ce n'est pas possible de trouver des codes préfixes dans des formes différentes, et un code préfixe peut être transformé à un autre on changeant les positions des 0s et 1s.

V. Unicité de décodage et théorèmes Kraft-McMillan [30]

Bien qu'en général, dans la littérature, on lise souvent le théorème Kraft-McMillan, il s'agit en fait d'un groupe de trois théorèmes. Ils sont dus à L. G. Kraft et B. McMillan Ces

théorèmes fournissent des conditions nécessaires et suffisantes pour s'assurer qu'un code donné soit décodable de façon non-ambigüe.

Théorème .1 : Théorème de Kraft (1). Soit C, un ensemble de n codes de longueurs l_1, l_2, ... ln exprimes en base b. Si l'ensemble de codes C est instantanément décodable (un code est instantanément décodable si, aussitôt le dernier symbole du code lu, on reconnait le code) alors les longueurs satisfont

$\sum_{k=1}^{n} b^{-l_k} \leq 1$ (*)

Théorème .2: Théorème de Kraft (2). Si l_1, l_2, . . ., ln et b sont tels que l'inégalité de l'´equation (*). Soit satisfaite, alors il existe un ensemble de codes C exprimés en base b qui est uniquement décodable.

La preuve du précédant théorème repose sur la construction explicite d'un code uniquement décodable qui satisfait obligatoirement l'inégalité de Kraft. Le théorème de McMillan est pour ainsi dire complémentaire aux deux théorèmes de Kraft. Le théorème de McMillan stipule que tout code uniquement décodable respecte nécessairement l'inégalité de Kraft.

Théorème .3 : Théorème de McMillan. Si C est un ensemble de codes uniquement décodables, alors les longueurs de codes l1, l2, . . ., ln satisfont obligatoirement l'inégalité de Kraft

L'importance des théorèmes Kraft-McMillan n'est pas à être sous-estimée. Ces théorèmes définissent des moyens pour tester la décodabilité unique d'un ensemble de codes en n'en connaissant que les longueurs, de même, cela permet de déterminer s'il existe en effet un ensemble de longueurs qui satisfasse l'inégalité. Puisque l'inégalité de Kraft est une condition nécessaire et suffisante à la décodabilité unique, il suffit de la vérifier pour déterminer la validité de la structure du code sous examen.

VI. Les types de codage entropique :

VI. 1. Le codage statistique

Le principe des algorithmes de codage statistique est d'utiliser les probabilités d'occurrence de chaque symbole dans une séquence de symboles émanant de la Source, dans ce cadre nous allons citer certains algorithmes comme **:**

VI. 1.1. Le codage Huffman [30]:

L'algorithme de Huffman produit des codes optimaux, dans la mesure où il produit des codes de longueur moyenne minimale, et bien qu'il puisse exister un certain nombre de codes Équivalents, aucune procédure ne peut produire des codes plus courts en moyenne sous les mêmes conditions, à savoir que les codes individuels sont de longueur variable mais entière et en faisant abstraction du coût de transmission de la description du code. La procédure de Huffman prévoit aussi le cas où nous avons plus de deux symboles de sortie : bien que nous ne soyons généralement concernés que par le cas binaire, ou l'alphabet de sortie est B = {0, 1}, nous pouvons nous trouver dans une situation où nous avons à notre disposition un nombre quelconque (mais toujours supérieur ou égal à deux) de symboles de sortie.

- ✓ **Le principe de la méthode :**

1. Les messages constituent les feuilles d'un arbre portant chacune un poids égal à la probabilité P d'occurrence du message correspondant
2. Joindre les 2 nœuds de moindre poids en un nœud parent auquel on attache un poids égal à la somme de ces 2 poids
3. Répéter le point 2 jusqu'à l'obtention d'une seule racine à l'arbre (de poids $\sum pi = 1$)
4. Affecter les codes 0 et 1 aux nœuds descendants directs de la racine
5. Continuer à descendre en affectant des codes à tous les nœuds, chaque paire de descendants recevant les codes L0 et L1 où L désigne le code associé au parent.

Par exemple, soit un ensemble de 3 messages a, b et c de probabilité respective { 0.6, 0.3 et 0.1}. La construction de l'algorithme est donnée à la figure 2.4

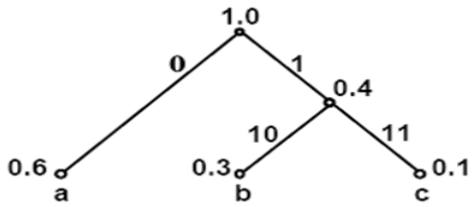


Fig 2.3: codage de Huffman

En codant des séquences de plus en plus longues, l'efficacité tend vers 100% (mais le gain est de moins en moins important, comme on le voit sur la figure 2.4).

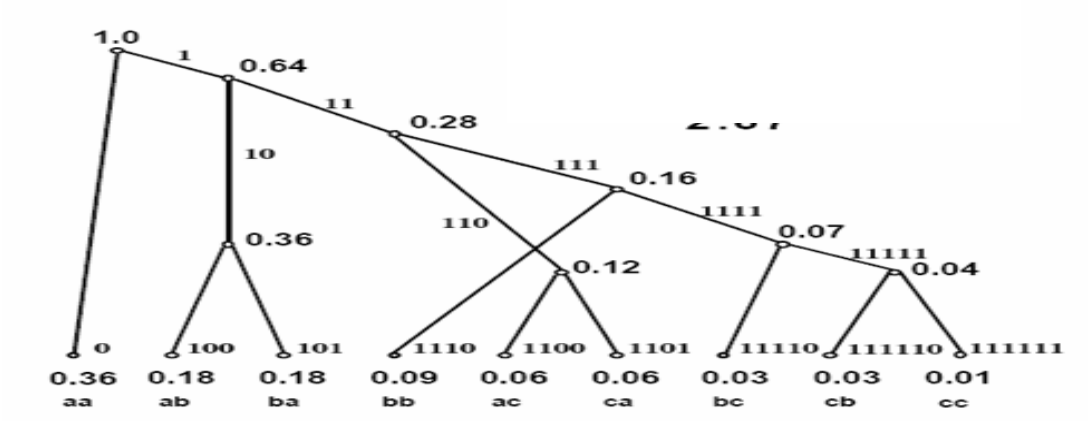


Fig 2.4: codage de Huffman 2

En résumé, l'algorithme de HUFFMAN donne un codage optimal (car la redondance est minimale) et possède la propriété préfixe (ce qui est intéressant en cas de transmission sur un canal).

- **Exemple2**

Considérons un fichier de 100 000 caractères que l'on souhaite stocker de façon compacte. On suppose que le nombre d'occurrences de chaque caractère est fourni comme suit :

Nb d'occurrence **:** a(45) b(13) c(12) d(16) e(9) f(5)

Par convention, on utilisera :

0=fils droit.

1=fils gauche

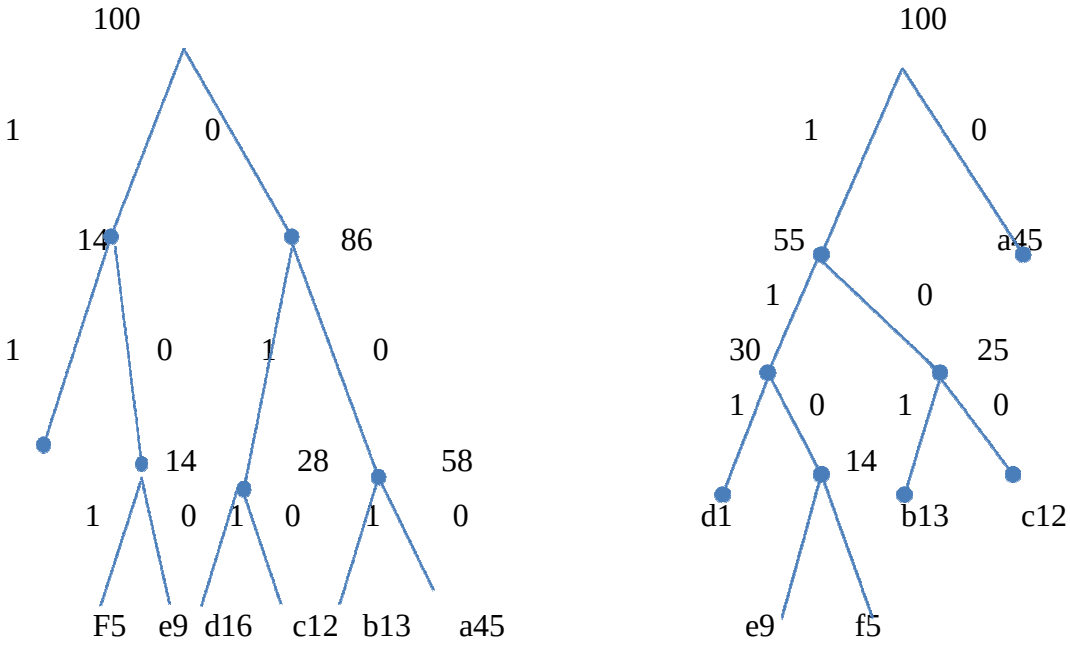


Fig2.5 : arbre binaire construit selon le codage de Huffman

- **Remarque**

Lors du classement des nœuds par probabilités décroissantes, il se peut que deux Nœuds aient mêmes probabilités. Leur classement est alors arbitraire. Lors de l'implantation des algorithmes, le choix le plus simple au niveau programmation est d'attribuer, en cas d'équiprobabilité, la place la plus élevée au dernier nœud créé. Nous adopterons cette coutume dans les exemples sachant qu'il s'agit ici d'un choix arbitraire.

VI. 1.2. Le codage de shannon-fano [30]:

Le procédé de Shannon-Fano construit un arbre descendant à partir de la racine, par divisions successives. Le classement des fréquences se fait par ordre décroissant, ce qui suppose une première lecture du fichier et la sauvegarde de l'en-tête.

- ✓ **Principe de la méthode**

- Classer les n fréquences non nulle {fi} par ordre décroissant
- Répartir la table des fréquences en deux sous table de fréquences proches
- Poursuivre l'arborescence jusqu'à ce que toutes les fréquences soient isolées .

- Attribuer dans l'arborescence le bit 0 à chaque premier sous table
- Attribuer aux symboles les codes binaires correspondant aux bits de description 1 de l'arborescence .

- **Un exemple de codage selon shannon_fanon** :

Nous allons effectuer la compression de notre message-témoin suivant « BANANES ET ANANAS » en utilisant l'algorithme de Shannon-Fano. Nous remarquons que, dans l'arbre de codage résultant, nous inscrivons un "0" en regard de la branche de gauche, et un "1" en regard de la branche de droite. Ce codage est sans effet sur l'efficacité du résultat, et est donc arbitraire.

La subdivision définit deux symboles (qui sont en fait des groupes de symboles), "AN" et ES<space>BT. Cette subdivision est ainsi faite pour que les fréquences des deux subdivisions Soient aussi équilibrées que possible. Voici la figure qui montre ces subdivisions :

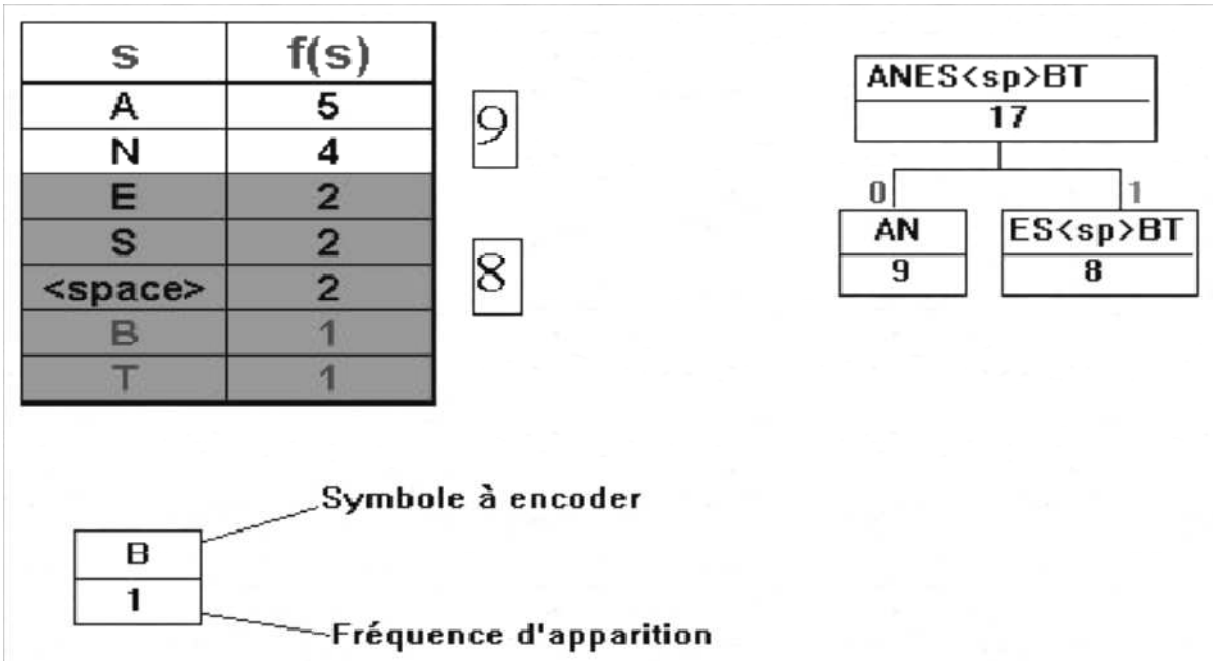


Fig2.6 : Les subdivisions de l'algorithme de Shanon-Fano.

La deuxième étape va isoler les symboles A et N, et définir une nouvelle subdivision des symboles restants.

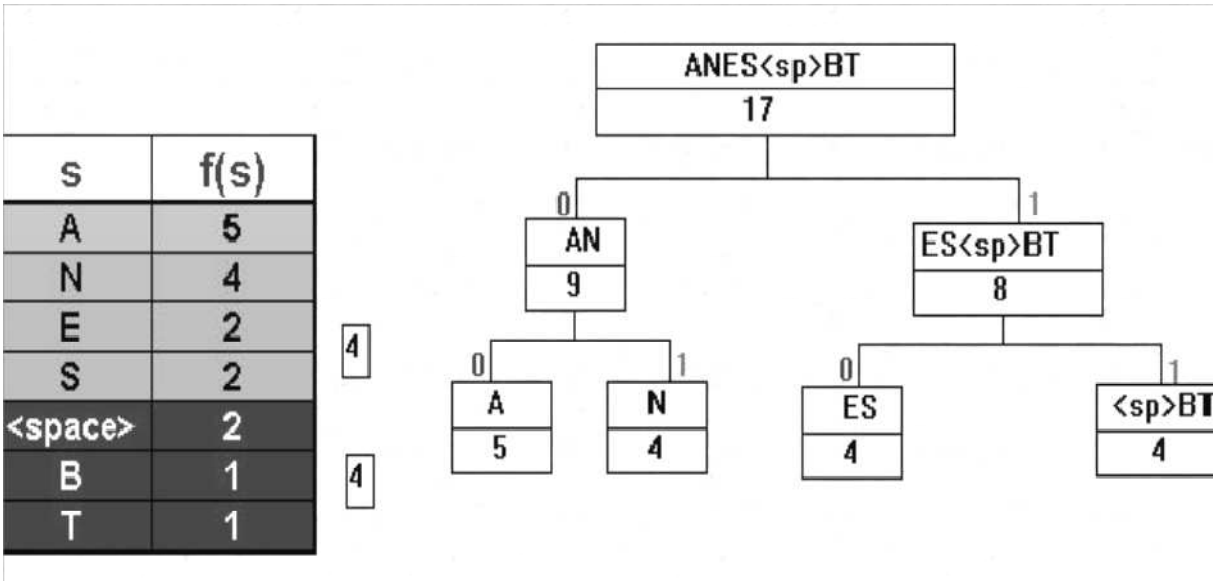


Fig2.7 : La 2ème étape de l'algorithme de Shannon-Fano

La troisième partie isole le ES et « <sp > BT » et ne laisse plus que le couple BT non résolu.:

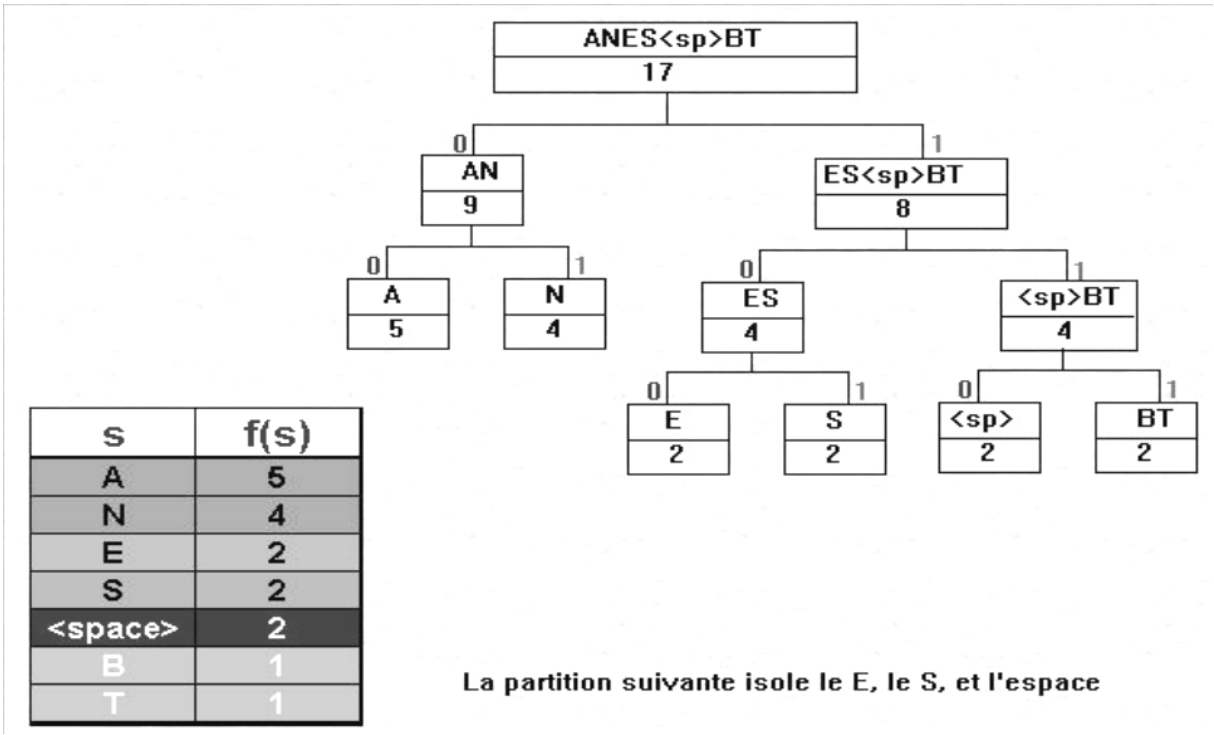
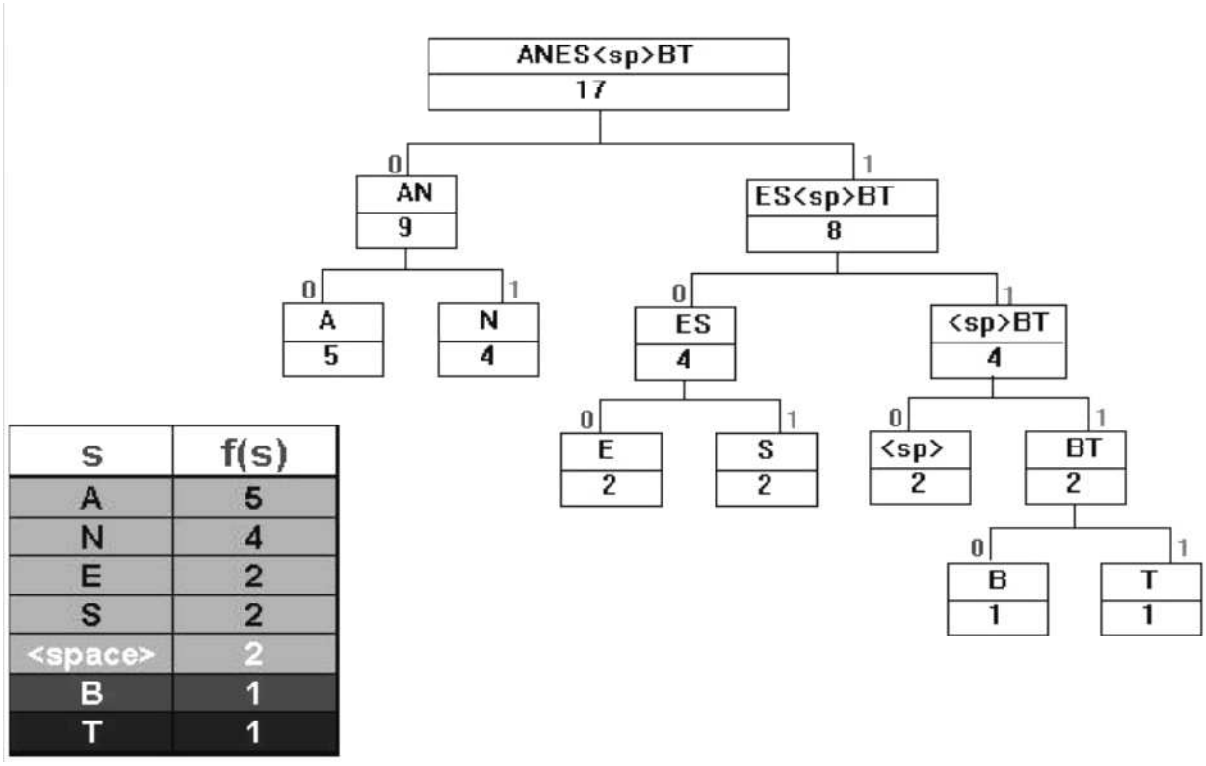


Fig2.8 : La troisième étape de l'algorithme Shanon-Fano.

Et la quatrième et dernière partie va isoler le couple « BT » et le résultat final est indiqué a la figure 1.6 :



s	f(s)
A	5
N	4
E	2
S	2
<space>	2
B	1
T	1

Fig2.9 La quatrième étape de l'algorithme Shanon-Fano.

Pour connaître le code associé à chaque lettre, nous parcourons l'arbre final de haut en bas, et nous obtenons :

A 00

N 01

E 100

S 101

<space>110

B 1110

T 1111

On peut s'assurer que le résultat correspond à une quantité de décision très proche de la quantité d'information, soit 44bits.la redondance résultant est pratiquement nulle.

La méthode serait lumineuse s'il n'était délicat de découvrir la méthode optimale. En pratique le codage shannon_fanon s'approche de l'optimisation idéale mais le risque existe de produire un code plus long que nécessaire.

- **Exemple2**

Pour créer un arbre de code en fonction de Shannon_Fano un tableau ordonné est nécessaire pour fournir les fréquences de n'importe quel symbole. Chaque partie de la table sera divisée en deux segments. L'algorithme doit s'assurer que l'un ou l'autre la partie supérieure et la plus inférieure du segment ont presque la même somme de fréquences.
Ce procédé sera répété jusqu'à ce que seulement des symboles simples soient laissés.

Symbole	Fréquence	longueur code	Le code	Total longueur
A	24	2	00	48
B	12	2	01	24
C	10	2	10	20
D	8	3	110	24
E	8	3	111	24

Tableau 2.2 exemple de shannon-fano.

Totale : 62 symboles codés avec Shannon_fanon :
Donc l'arbre est donné comme suit :

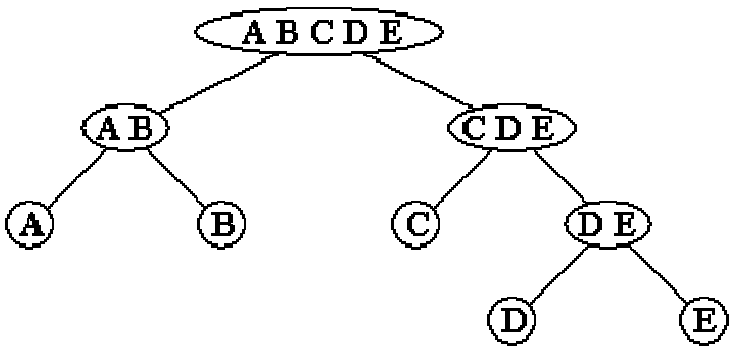


Fig 2.10:construction d'un arbre selon le code shannon fanon

Les données originales peuvent être codées avec une longueur moyenne du bit 2,26bit. Avant de produire un arbre de code de Shannon-Fano la table doit être connue ou elle doit être dérivée des données précédentes.

- **Remarque sur les deux algorithmes :**

En principe, le codage de Huffman donne toujours le meilleur résultat possible en termes de compression. Avec ces algorithmes, les données compressées doivent être accompagnées de la table ou arbre de codage. La compression n'est rentable que si la représentation des deux : données et arbre est plus petite que les données d'origine. On peut imaginer que pour des données similaires, par exemple des articles de journaux en français, on puisse avoir la même table de codage basée non pas sur les statistiques d'apparition des lettres et non pas une table de codage par article. Nous y perdrions sur certains articles mais nous éviterions de stocker une table dans chaque article. Ces deux méthodes travaillent sur des données totalement aléatoires

VI. 2. Le code unaire[31]:

Le code à longueur variable le plus simple et le plus intuitif à lequel on peut penser est le code unaire.

Le code unaire pour entier positif N est constitué d'une suite de N bits à '1' suivis d'un bit à 0 ou bien le code inverse avec une suite de N bits à '0'suivis d'un bit à '1' .la taille du code pour chaque entier N est (N+1)bits ,le tableau suivant illustre le code unaire.

N	Code	Code inverse
0	0	1
1	10	01
2	110	001
3	1110	0001
4	11110	00001
5	111110	000001
…	…	…
i	(111111…….10)	(000000……..01)

Tableau 2.3 code unaire

On peut facilement remarquer que le code unaire satisfait la propriété de préfixe. L'utilisation de ce code dans la compression se fait en attribuant le code '0' au symbole qui possède la plus grande probabilité.

VI. 3. Le codage arithmétique [30] :

Le codage arithmétique est la méthode la plus efficace pour coder des symboles selon la probabilité de leur occurrence. La longueur moyenne de code correspond exactement au minimum possible indiqué par théorie de l'information. Les déviations qui sont provoquées par la peu-résolution des arbres de code binaire n'existent pas.

Contrairement à un arbre binaire de code Huffman le codage arithmétique offre un taux de compression nettement meilleure de compression. Son exécution est plus complexe d' autre part.

Le Codage arithmétique fait partie du format de données JPEG. Alternative au codage de Huffman il sera utilisé pour le codage final d'entropie.

- ✓ **Principe du codage :**

- Calculer la probabilité associée à chaque symbole dans la chaîne à coder.
- Associer à chaque symbole un sous intervalle proportionnel à sa probabilité, dans l'intervalle [0,1] (l'ordre de rangement des intervalles sera mémorisé car il est nécessaire au décodeur).
- Initialiser la limite inférieure de l'intervalle de travail à la valeur 0 et la limite supérieure à la valeur 1.
- Tant qu'il reste un symbole dans la chaîne à coder :

  largeur = limite supérieure - limite inférieure.

  limite inférieure = limite inférieure + largeur x (limite basse du sous intervalle du symbole).

  limite supérieure = limite inférieure + largeur x (limite haute du sous intervalle du symbole).
- La limite inférieure code la chaîne de manière unique que le fichier dont il est redondant.

Ces chiffres décimaux dépendent non seulement des symboles du fichier dans l'ordre où ils apparaissent, mais aussi de leur distribution statistique.

✓ **Exemple de codage arithmétique:**

On considère la source S= {a, b, c, d, e} avec les probabilités respectives d'occurrence des symboles suivantes : P(a)=0.3, P(b)=0.25,P(c)=0.20, P(d)=0.15, P(e)=0.1

On veut coder la séquence bdcea, Pour coder cette séquence on divise l'intervalle [0,1[en 5 sous-intervalles, puis on se place sur le sous-intervalle correspondant au premier symbole de la séquence à coder, il s'agit du symbole "b". Pour le symbole suivant de la séquence "d" on Subdivise le sous intervalle de b, [0.3, 0.55[en 5 sous-intervalles correspondant au nombre de symboles de l'alphabet de la source S.

On procède ainsi récursivement pour toute la séquence (voir figure ci-dessous).

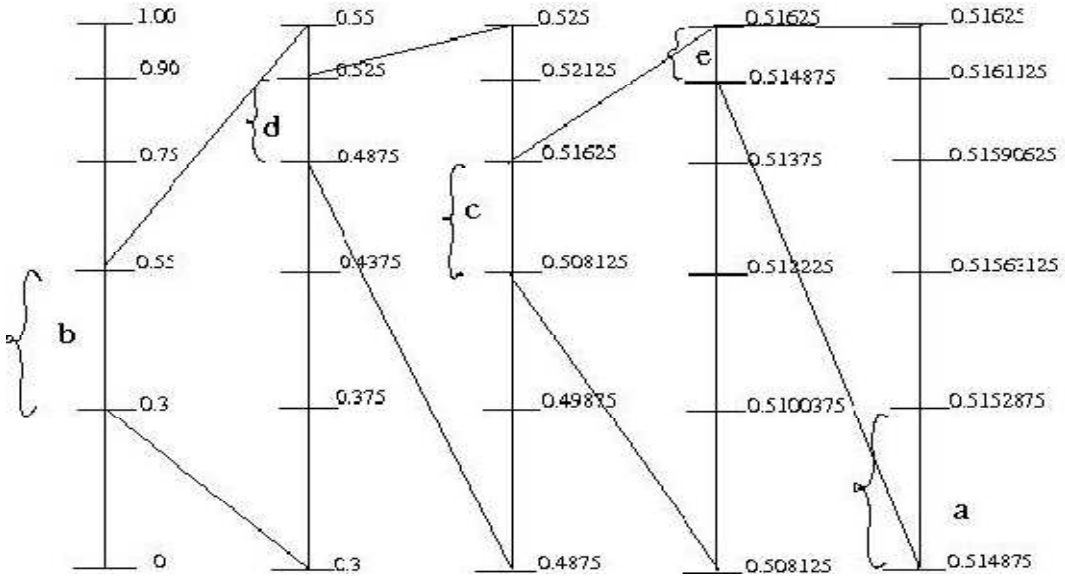


Fig 2.11: exemple de codage arithmétique

✓ **Décodage**

Cet algorithme comporte six étapes successives :

1) On initialise a_c=0 et b_c=1

2) On calcule la largeur du sous-intervalle du code : largeur = b_c-a_c

3) On trouve le sous-intervalle [a_k , b_k[du symbole s_k avec $1 \leq k \leq N$ tel que :

$a_k \leq \frac{(x_c - a_c)}{\text{largeur}} \leq b_k$, Sachant que x_c est le réel codant la séquence.

4) On obtient le symbole s_k

5) On met à jour le sous-intervalle de codage : a_c=a_c+largeur x a_k et b_c=a_c+largeur x b_k

6) On répète les 2, 3,4 et 5 jusqu'à obtenir le décodage de tous les symboles de la séquence.

✓ **Exemple de décodage :**

On applique l'algorithme de décodage à l'exemple précédent :

On considère la valeur x_c=0.51508125 codant la séquence

Etape 1 : On initialise a_c=0 et b_c=1

Etape 2 : On calcule la largeur du sous-intervalle du code : largeur = b_c-a_c =1

Etape 3 : On calcule le nombre $\frac{x_c - a_c}{\text{Largeur}}$ dont la valeur est 0.51508125 et on cherche k tel que ce nombre soit compris dans la partition initiale.

Etape 4 : k=2, Il s'agit du sous-intervalle [0.3, 0.55[qui correspond au symbole b.

Etape 5 : On met à jour le sous-intervalle de codage $a_c=a_c$+largeur x a_k et $b_c=a_c$+largeur x b_k

a_c=0 + 1x0.3=0.3

b_c=0+1 x 0.55=0.55

On répète l'étape 2 : largeur = 0.55-0.3=0.25

Etape 3 : (0.51508125-0.3)/0.25 = 0,860325

Etape 4 : k=4, il s'agit du sous-intervalle [0.75, 0.90[qui correspond au symbole d.

On revient à l'étape 5 et ainsi de suite 8

VI. 3.1. Code d'Elias[30] :

Elias a eu l'idée d'encoder complètement récursivement la longueur du code. Puisque la longueur est codée récursivement, il faut commencer par lire les premiers bits. Le premier bloc est de longueur 2. Un bloc de bits qui commence par un zéro détermine la fin du code, et le bloc précédent contient l'entier n à lire. Si on lit les zéro dés le premier coup, le nombre est n = 1. Si le code commence plutôt par un, alors on lit le nombre de bits (moins un, puisque nous avons déjà lu le premier bit) indiqué par le bloc précédent. Ce nombre de bit indique la taille du prochain bloc, et on poursuit jusqu'à ce que l'on trouve un premier bit à zéro.

Le code pour n est donné par :

$$C\omega(n) = \begin{cases} 0 & \text{si } n = 1 \\ H\omega(n) : 0 & \text{si } n > 1 \end{cases}$$

Où :

$$H\omega(n) = \begin{cases} 0 & \text{si } n = 1 \\ 10 & \text{si } n = 2 \\ 11 & \text{si } n = 3 \\ H\omega([\lg n]) : C_\beta(n) & \text{si } n > 4 \end{cases}$$

Le code généré par cette formule est donné dans le tableau suivant :

1	0
2	10 0
3	11 0
4	10 100 0
5	10 101 0
6	10 110 0
7	10 111 0
8	11 1000 0
...	
...	
14	11 1110 0
15	11 1111 0
16	10 100 10000 0
17	10 100 10001 0
…	

Tableau 2.4 Code d'Elias pour les entiers

On distingue dans le code Elias trois codes qui sont définit comme suit :

- **Code gamma[31] :**

Le codage gamma ou codage gamma d'Elias est un codage entropique inventé par Peter Elias et utilisé essentiellement en compression de données.et permet de coder tous les entiers naturels, à l'exception de zéro, sans qu'il y ait besoin de connaitre au préalable l'intervalle des

valeurs à coder (contrairement, par exemple, au codage binaire de taille fixe, qui ne permet de coder que des nombres inférieurs à une borne supérieure fixée à l'avance).

- ✓ **Le codage :**

1**.** chercher le plus grand nombre M tel que $2^M \leq N < 2^M+1$,$N=2^M+L$ avec la longueur de Lest au plus Mbits .

2. coder le nombre M aves le code unaire M'0' suivi par '1' ou M'1' suivi par '0'

3. concaténer les deux codes M.L+.

Exemple :

$1 = 2^0 + 0 = 1$ (M=0, L=0)	$9 = 2^3 + 1 = 0001\ 001$
$2 = 2^1 + 0 = 01\ 0$ (M=1, L=0)	$10 = 2^3 + 2 = 0001\ 010$
$3 = 2^1 + 1 = 01\ 1$ (M=1, L=1)	$11 = 2^3 + 3 = 0001\ 010$
$4 = 2^2 + 0 = 001\ 00$	$12 = 2^3 + 4 = 0001\ 100$
$5 = 2^2 + 1 = 001\ 01$	$13 = 2^3 + 5 = 0001\ 101$
$6 = 2^2 + 2 = 001\ 10$	$14 = 2^3 + 6 = 0001\ 110$
$7 = 2^2 + 3 = 001\ 11$	$15 = 2^3 + 7 = 0001\ 111$
$8 = 2^2 + 0 = 0001\ 000$	$16 = 2^4 + 0 = 00001\ 0000$

Tableau 2.5 code Gamma .

- ✓ **Décodage :**

Pour le processus de la décompression il est aussi simple et se fait seulement en deux étapes :

- lire les zéros '0'depuis le code jusqu'à rencontrer le bit '1', trouver le nombre de '0' et noter M.
- lire les M bits comme un entier L, et calculer $N=2^M+L$

il est facile de voir que ce code peut être employé pour coder des nombres entiers positifs même dans les cas où le plus grand nombre entier n'est pas connu à l'avance .en outre ,ce code se développe lentement ,ainsi c'est un bon candidat pour la compression des nombres entier ou les petits nombres entiers sont communs et le grand est rare.

- ➢ **Code Delta[31] :**

Elias a proposé un autre algorithme de construction d'un code VLC qui est le code***δ*** (delta).ce code peut se construire de deux manière déférentes :

1-par l'ajout des longueurs en binaire.

2-ou par l'emploi de code Gamma précédent

Ainsi, le code Delta d'Elias, aussi pour les nombres entiers positifs, est légèrement plus complexepour le processus de construction.

- ✓ **Le Codage**

Pour le première cas, Ajout de longueurs en binaire :

Dans son code gamma, Elias ajoute la longueur du code unaire (α). Dans son prochain code, δ (delta), il ajoute la longueur en binaire (β) Alors la construction de cet algorithme se déroule comme suit :

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

Exemple 1 : Quand ces étapes sont appliquées au nombre entier 17, voici les résultats :

Ecrire le nombre 17 en binaire : 17 = 10001 (5bits).

Enlever le 1 le plus à gauche et ajouter 5 = 101 nous aurons : 101|0001.

Trois bits étaient supplémentaires, ainsi nous ajoutons deux zéros pour obtenir le code de delta :00|101|0001.

Pour le deuxième cas, Avec l'emploi de code Gamma : l'algorithme est fait selon les étapes suivantes :

1. Chercher le plus grand nombre M tel que, $2^M \leq N < 2^{M+1}$, $N=2^M$ +L avec la longueur de L est au plus M bits.
2. Coder le nombre M+1 avec le code gamma d'Elias.
3. Concaténer ensuite les deux codes M.L

Exemple 2:

N=17, le résultat est 17= 24+1. Le code gamma de M+1=5 est 00101 puis concaténer avec L=0001 pour donner N=00101 000.

Le tableau suivant illustre les codes Delta pour les 16 premiers entiers:

$1=2^0+0=1$ (M=0, L=0)	$9=2^3+1=00100\ 001$
$2=2^1+0=010\ 0$ (M=1, L=0)	$10=2^3+2=00100\ 010$
$3=2^1+1=010\ 1$ (M=1, L=1)	$11=2^3+3=00100\ 011$
$4=2^2+0=011\ 00$	$12=2^3+4=00100\ 100$
$5=2^2+1=011\ 01$	$13=2+5=00100\ 101$
$6=2^2+2=011\ 10$	$14=2^3+6=00100\ 110$
$7=2^2+3=011\ 11$	$15=2^3+7=00100\ 111$
$8=2^3+0=00100\ 000$	$16=2^4+0=00101\ 0000$

Tableau 2.6 Exemple de codage Delta.

- ✓ **Le décodage :**

Le décodage est fait selon les étapes suivantes :

1-lire la chaine de gauche à droite jusqu'à reconnaitre un code gamma qui représente M+1.

2-lire les M bits suivants qui représente le nombre L.

3-calculer $N=2^M+L$.

- ➢ **Le code Omega[31] :**

Comme le codage gamma et le codage delta, le codage Omega permet de coder tous les entiers naturels, à l'exception de zéro, sans qu'il y ait besoin de connaitre au préalable l'intervalle des valeurs à coder (contrairement, par exemple, au codage binaire).

Pour cela, le codage Omega fait précéder la représentation binaire du nombre par sa longueur (en nombre de bits), comme le font les codages précédents. Comme l'intervalle des valeurs pouvant être prise par cette longueur n'est pas non plus connu à l'avance, elle doit être codée avec un code préfix. Contrairement au codage gamma qui utilise un codage unaire à cet

effet, ou au codage delta qui utilise un codage gamma, le codage Omega utilise un codage Omega. Ce codage est donc un codage récursif.

- ✓ **Le Codage**

Le codage d'un nombre entier positif N est fait périodiquement dans les étapes suivantes :

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 1 : Le nombre entier 17 par exemple est donc codé par :

1) un 0 simple,
2) ajouté au début, la représentation binaire de 17 qui est 10001 sur 5 bits,
3) ajouté au début la représentation binaire de 5−1 qui est 100 sur 3 bits,
4) ajouté au début la représentation binaire de 3−1 qui est 10 sur 2 bits,
5) arrêté car 2−1 = 1 Le résultat est 10|100|10001|0.

Le tableau suivant énumère les 16 premiers codes d'Omega :

1= 0	9= 11 1001 0
2=10 0	10=11 1010 0
3= 11 0	11=11 1011 0
4= 10 100 0	12= 11 1100 0
5= 10 101 0	13= 11 1101 0
6= 10 110 0	14= 11 1110 0
7= 10 111 0	15= 11 1111 0
8= 11 1000 0	16= 10 100 10000 0

Tableau 2.7 Exemple de codage Omega.

- ✓ **Le décodage :**

Le décodage est fait dans 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 2: Décoder 10|100|10001|0.

Le décodeur initialise N = 1 et lit le premier bit. C'est un 1, ainsi il lit N = 1 bit de plus (0) et assigne N = 102 = 2. Il lit le prochain bit. C'est un 1, ainsi il lit N = 2 bits de plus (00) et assigne le groupe 100 à N. Il lit le prochain bit. C'est un 1, ainsi il lit quatre bits de plus (0001) et assigne le groupe 10001 à N. Le prochain bit lu est 0, indiquant la fin du décodage. Donc la valeur de N est N= 100012 = 17.

- ✓ **Taille des codes d'Elias :**

Le tableau suivant montre la taille des différents codes (Gamma ,Delta et Omega) :

Valeur	Code Gamma	Code Delta	Code Omega
1	1	1	1
2	3	4	3
3	3	4	3
4	5	5	6
5-7	5	5	6
8-15	7	8	7
16-31	9	9	11
32-63	11	10	12
64-127	13	11	13
1000	19	16	17
10^4	27	20	21
10^5	33	25	28

Tableau 2.8 taille des trois codes d'Elias

VI. 3.2. Le codage de Golomb[32] :

Le codage de Golomb est un codage entropique inventé par Solomon Wolf Golomb en

1966 et utilisé essentiellement en compression de données. Le code produit est un code préfix.

- ✓ **Encodage :**

Le codage de Golomb pour les entiers non négatifs « n » dépend du choix d'un paramètre « m ».

c'est un code préfixe paramétré, ce qui fait de lui un code spécialement utilisable dans les cas où des bonnes valeurs pour le paramètre peuvent être calculés ou estimés .la première étape de construction d'un code de Golomb pour un entier non négatif « n » est de généré les trois quantités « q » (quotient), « r »(reste) ,et « c » tel que :

q= [n/m], r= n-qm, et c= [$\log_2$(m)]

Le code est construit par deux parties :

-la première est la valeur de « q », codé en code unaire, et le deuxième est la valeur binaire de « r » codé d'une manière spéciale. Les premiers 2^c-m valeurs de « r » sont codé comme des nombres non signé sur c-1bits pour chacun. et le reste des valeurs de « r » sont codé sur « c »bits chacun (le mot finale est une suite de c 1s) .le cas où m est une puissance de 2($m=2^c$) est spéciale parce que il n'y a pas des mots de c-1 bits .on sait que **n = r+qm** donc une fois que le code Golomb est décodé peut être facilement utilisé pour reconstruire n. Le cas m=1 est aussi un cas spécial .dans ce cas q=n et r=c=0, implique que le code Golomb de n est uniquement un code unaire.

- ✓ **Décodage :**

Les codes Golomb sont désignés d'une manière spéciale pour faciliter leur décodage. on commence par démonter le décodage pour un cas simple m=16 (m est une puissance de 2). Pour décoder on commence par l'extrémité gauche du code et on compte le nombre A de 1s avant le premier 0 .la longueur du code est A+c+1 bits (pour m=16, on a A+5 bits) .si on dénote les 5 bits qui sont sur la droite, par R, alors la valeur de code est 16A+R. ce simple décodage montre le chemin comment le code est construit .pour encoder n avec m =16 ,on commence par diviser n par 16 pour avoir n=16A+R ,ensuite on écrit A1s suivis d'un 0 singulier ,suivi par 4bits qui représente R.

Pour des valeurs de m qui ne sont pas des puissances de 2. Le décodage est un peu plus compliquer. On assume que le code commence par A 1s, on commence par enlever avec le 0 qui les suit. on dénote les c-1 bits qui suivent par R. si $R<2^c$-m, alors la longueur du code est

A+1+(c-1) (les A 1s , le zéro qui le suit ,et les c-1 bits qui suivent) et la valeur du mot est m*A+R. si $R \geq 2^c$-m, alors la longueur total du code est A+1+c et ça valeur est m*A+R'-(2^c-m),où R' est le nombre sur c bits qui constitué de R et le bit qui suit le R.

VII. Le codage optimal [33] :

- **Définition**

soit un alphabet S=(s1,s2,…..sn) , et la distribution de probabilité p = (p_1,p_2……..,p_n) de chaque symbole . supposons que nous dérivons un code C= (c_1,c_2,……,c_n) avec longueur moyenne de chaque mot de code L=(l_1,l_2,…..,l_n). La longueur moyenne d'un mot de code est

$$L=\sum_{i=1,\ldots n} l_i p_i$$

Remarquons qu'en toute rigueur cette longueur moyenne dépend du codage considéré et la distribution de probabilité de la source ; on devrait donc utilisé une notation du type lc,p. Cependant, on utilise en général une notation allégée lorsqu'il n'y a pas d'ambiguïté au sujet du codage ou de la distribution de probabilités considérées.

- **Exemple1 :**

Soit la source S = (s1,s2,….,sn) ,avec la distribution de probabilités P=(0.3,0.3,0.25,0.15).
On a de manière générale pour un codage C de longueur (l1, l2, l3, l4) :

L = 0.3l1 +0.3l2 +0.25l3 +0.15l4.

On prend maintenant 3 code C1=(1,00,01,10) ,C2= (0,11,100,101) et C3 = (00,01,001,110).comparons les longueurs moyennes qui leur sont associées :

Lc1 = 0.3*1+0.3*2+0.25*2+0.15*2=1.7

Lc2 =0.3*1+0.3*2+0.25*3+0.15*3=2.1

Lc3 =0.3*2+0.3*2+0.25*3+0.15*3=2.4

Donc on moyenne. Le codage d'une information avec C1 est plus court que le codage C2 qui lui-même plus court que le codage avec C3.

- **Définition 2**

un code C est optimal si son efficacité est maximale .par d'autre mots un code est optimal si la longueur moyenne de ces mots est égale à l'entropie de la source .

- **Proposition :** la longueur des codes des lettres dans un code optimale C est une fonction décroissante de leur probabilité d'occurrence :

$$x, y \in S, P(x) > P(y) \rightarrow L(x) \leq L(y)$$

VIII. Construction de code optimal :

✓ Cas de distribution uniforme :

Soit N le nombre de symbole de la source , on part d'un code de longueur fixe minimale q avec $2^{q-1} < N < 2^q$ si $N=2^q$, alors on ne peut pas faire mieux .par contre si $N<2^q$,on peut remplacer 2^q –N mots par des mots de longueur q-1.

✓ Cas de distribution de la forme $P(x)=1/2^k$

Soit S =(s1,…,sn) la source à coder ,et $pi=P(s_i)$ les probabilités associées aux symboles .on suppose dans ce paragraphe que ces probabilités sont de la forme $1/2^k$, avec $k \geq 1$, et que les symboles sont ordonnées de manière à avoir $p1 \geq p2 \geq \ldots \geq pn$. Rappelons que par définition on a $\sum_i pi = 1$.

La proposition suivante établit qu'à partir $x \in s$ de tout codage optimal d'une source ayant au moin deux symboles .on peut découper cette source en deux et extraire du codage optimal deux codages optimaux pour chacun des deux morceaux de la source.

- **Proposition :**

Soit C un code préfixe optimal pour la distribution de probabilité P . les sous_arbres gauche et droit induisent des codages optimaux C1 et C2 pour les sources S1(des lettre dont le code commence par un 0) et S2 (des lettres dont le code commence par un 1) , avec les distributions P1 et P2 qui sont les restriction respectives de P à S1 et S2 :

$$Pi(s) = \frac{p(s)}{\sum x \epsilon si\ P(x)} \text{ Pour i=1,2}$$

Voici une méthode proposée R.M.Fano pour construire un code optimal :

-si N=1, alors on associe le mot vide au symbole.

- si N=2, alors on a forcément P1 =P2 = ½ et le code optimal est le code de 2 mots de longueur 1 : au premier symbole on associe la lettre 0, et au deuxième la lettre 1.

- si N>2, on procède récursivement :

1. on peut partitionner la source S en deux parties S1 et S2.

2. pour chacune de ces deux parties, on réapplique séparément la procédure en pondérant les probabilités comme indiqué à la proposition précédente.

3. on ajoute la lettre 0 en tête des mots de codes associés aux symboles de S1, et la lettre 1 en tête de mot de code associer aux symboles de S2

- ✓ en général on peut utiliser la méthode de Huffman pour construire le code optimal.

IX. Simplification de la table de VLC

Une méthode permettant de réduire la complexité de décodage consiste à utiliser une table de mots de code de taille réduite. Nous présentons ici un algorithme de regroupement des mots de code de même longueur en classes. Ainsi, par exemple, les 16 mots de code de 8 bits utilisés pour le codage de la texture dans la norme H.263+
A8 = {00100000, 00100001 . . .00101110, 00101111}
sont formés du préfixe 0010 et de tous les suffixes possible de 4 bits. A8 peut ainsi être décrit de manière compacte par 0010$$$$, où $ représente soit 0, soit 1. A l'aide de cette simplification, dans le treillis présenté au paragraphe précédente, 16 branches parallèles entre les noeuds (a) et (a + 8) peuvent être remplacées par une branche unique. Le principe de l'algorithme proposé généralise la technique de réduction présentée précédemment.

X. Conclusion :

Dans ce chapitre nous avons présentés quelques définitions concernant la méthode de code à longueur variable (VLC, et aussi appelé codage entropique) qui sont utiles pour la compression de données, ainsi nous avons entamées tous les point importants de cette méthodes comme les divers techniques et algorithmes accompagnes de plusieurs exemples.

On remarque que les techniques et les approches VLC sont très importantes et très efficaces vu le rôle qu'elles jouent, et le principe sur lequel sont basées qui consiste à remplacer les symboles les plus fréquents par des petits codes et vice versa.

CHAPITRE 3 :

ANALYSE ET CONCEPTION

I. Introduction

Dans Ce chapitre nous allons présenter en premier lieu la nouvelle méthode de compression de données basée sur VLC, nous détaillerons par la suite le code proposé ainsi que ses variantes.

Ensuite nous allons fixer notre objectif et présenter l'architecture générale de notre application ainsi que ses différents modules et sous modules en expliquant le processus de codage et de décodage. Ce système doit fournir en plus des fonctionnalités de compression/décompression d'autres fonctions telles que le calcul de temps et le taux de compression

Enfin nous allons présenter les différents algorithmes nécessaires a la mise en œuvre de notre l'application.

II. Présentation générale de la méthode proposée

La méthode de compression à longueur variable proposée dans ce projet est une méthode de compression sans perte, où lors de la décompression on aura le même fichier d'origine.

La compression de données par l'approche VLC est effectuer en affectant des codes préfixés ayant les plus petites longueurs possible pour les symboles ayant les plus grandes probabilités P(Si), avec Si est un symbole du fichier source, comme le montre le tableau suivant :

Probabilité du symbole [p(Si)]	Le code VLC [C(Si)]
P_0	C_0
P_1	C_1
.	.
.	.
.	.
P_i	C_i

Avec $P_0 \geq P_1 \geq \ldots \geq P_i$ et $L(C_0) \leq L(C_1) \leq \ldots \leq L(C_i)$, tel que $L(C_i)$: longueur du code C_i

Tableau 3.1 Principe générale de la méthode proposé.

II. 1. Le code proposé

Le code proposé est un code qui est constitué de deux parties, une partie fixe (preffix) et une partie variable (suffix),

II. 1.1. Le codage

Pour un codage fixe sur N bits on aura 2^N symboles S= {S_0, S_1, … S_{2^n-1}}.

Le codage VLC correspondant à cet ensemble de symboles est :

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

II. 1.1.1. Le Preffix : ce champ représente deux (02) informations :

1. Il indique la position (P) du bit le plus significatif dans le code fixe de Si. Sa valeur Val (*preffix*) = P∈ [0, N−1], (*Cas particulier* : Pour S_i=0→preffix =0)
2. Il indique la longueur de la partie *suffix.*

- **La longueur de la partie préfix**e

C'est la taille en nombre de bits du champ *preffix*.

Le nombre de bits, n, nécessaire pour représenter un nombre X est :

$$n = 1+\log_2 X \approx \log_2(X+1) \ldots..(1)$$

- **Exemple** : le nombre de bits pour représenter la valeur x=8 est :

$$\text{Log}_2(8)+1= \frac{\log(8)}{\log(2)} \approx 1+3 = 4 \text{ bits.}$$

Pour calculer la longueur du champ *preffix*, on doit trouver la taille da la plus grande valeur du *preffix* appartenant a l'intervalle [0, N-1] qui est X=N-1.

Appliquant la règle (1) :

L (*preffix*)= L(N-1)= $\log_2((N-1)+1)= \log_2(N)$ bits.

L (*preffix*) : longueur du *preffix* en nombre de bits.

II. 1.1.2. Le Suffix: c'est la partie restante du code fixe S_i qui suit le bit significatif.

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

- **Exemple 1** : N=8 bits, le nombre de symboles est $2^N = 2^8 = 256$.

$S_4 = 4_{(10)} = \underbrace{(00\ldots\ldots100)_2}_{\textbf{N bits}}$, de la on a :

- ✓ Val (*preffix*) = P = 2 (la position du bit le plus significatif de la droite à gauche).
- ✓ Val (*suffix*) = 00.
- ✓ L (*suffix*) = *preffix* = 2bits.
- ✓ le code VLC correspondant au symbole S4 = $(00\ldots100)_2$ est $C(S_4)$ $C(S_4)$ = preffix $_4$ .suffix $_4$ = $\underbrace{010}_{\textit{Preffix}}\ \underbrace{\mathbf{00}}_{\textit{suffix}}$ = 01000

- **Exemple 2 :** N=4 bits, S_i=1 jusqu'à 15 l'ensemble des symboles.

Compression de la chaîne : CH=$S_1S_1S_1S_2S_3$ S_3. La représentation binaire de CH est 0001 0001 0001 0010 0011 0011, la taille de CH est: L(CH)=4*6=24 bits.

La compression consiste à remplacer chaque code de taille fixe 4 bits par Son code VLC correspondant suivant les étapes :

i. Calculer les probabilités des symboles et les triés selon l'ordre décroissant :

Indice	Symbole	Probabilité $P(S_i)$
0	S_1	0.5
1	S_3	0.33
2	S_2	0.16

Tableau 3.2 les probabilités des symboles et leur tris de l'exemple 2.

ii. Construire le code VLC

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

$C(S_i) = preffix_i . suffix_i$

N=4bits →L($preffix_i$)=$\log_2(4)$=2bits.

Indice	Indice binaire	Position du bite poids fort	Code $C(S_i)$	
			preffix	suffix
0	0000	0	00	0
1	0001	0	00	1
2	0010	1	01	0

Tableau 3.3 construction des codes VLC de l'exemple 2.

iii. Remplacer chaque code fixe par son code préfixé.

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

CHc : chaine compressée. L(CHc) = 3x6 =18 bits.

iv. La moyenne de bits par symbole :

$$\sum_{i=1}^{N} P(Si)L(Ci) = 0.5 \text{ X } 3 + 0.33 \text{ X } 3 + 0.16 \text{ X } 3 = 2.99$$

$$\sum_{i=1}^{N} P(Si)\ L(Ci) \approx 3bits\ /symbole$$

II. 1.2. Le décodage :

Pour le processus de décodage, nous devons connaitre la taille du code initial (N) et la table des probabilités des différents symboles puis procéder à la décompression suivant les étapes :

- Lire Log_2 N premier bits (*preffix*).
- Si *preffix* = 0, lire 1 bit.

  Sinon lire *preffix* bits (*suffix*).

- Calculer indice $=2^{\text{preffix}} +$ *suffix*.
- S*i* = Table [indice] et le représenter sur N bits.

II. 1.3. La longueur du code

La longueur de chaque code VLC assigné aux codes fixe sur N bits est calculée suivant la relation : $L(C_i) = L(preffix_i \,.\, suffix_i) = L(preffix_i) + (suffix_i) = \log_2(N) + preffix_i$

Nous avons : L (suffix i) $= \begin{cases} 1 \ si \ preffix \ i = 0 \\ preffix \quad sinon \end{cases}$

Donc : $L(C_i) = \begin{cases} \log 2(N) + 1\,, & i = 0 \\ \log 2(N) + \ preffix \ i\,, \forall\, i \in [1, 2^N - 1] \end{cases}$ ……(2)

$L(Ci) = Cte + preffix_i$, avec $Cte = \log_2(N)$

Et sachant que : *preffix* i ∈ [0, N-1] → Le nombre de taille du codes est N différentes tailles, Alors (2) devient :

$$L(Ci) = \begin{cases} \log 2(N) + 1, & i = 0 \\ \log 2(N) + prefix\ i, & prefix\ i \in [1, N-1] \end{cases}$$

- ✓ **Remarque:**

Pour un ensemble de symbole sur N bits, on a les résultats suivants :

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

- **Exemple 3 :**

Supposons n= 8 bits, il y a 2^8 =256 symboles. Construisons le code VLC correspondant.

$C(S_i) = prefix_i . suffix_i, \forall i \in [0, 2^8 - 1]$

- Calculer le nombre de bits de la partie préfixe :

$L(prefix_i) = \log_2(n) = \log_2(8) = \frac{\log(8)}{log2} = 3$bits.

- Le tableau suivant illustre les différents codes

Indice	Symbole Si	Bit poids fort	Code Ci		L(Ci) Log2(8)+p	Nombre de code $[2^{L(suffix)}]$
			preffix	*suffix*		
0 -1	0000000X	0	000	X	4	2
2-3	0000001X	1	001	X	4	21 =2
4-7	000001XX	2	010	XX	5	22 =4
8-15	00001XXX	3	011	XXX	6	23 =8
16-31	0001XXXX	4	100	XXXX	7	24 =16
32-63	001XXXXX	5	101	XXXXX	8	25 =32
64-127	01XXXXXX	6	110	XXXXXX	9	26 = 64
128-255	1XXXXXXX	7	111	XXXXXXX	10	27 =128

Tableau 3.4 Liste des différents codes VLC.

✓ **Remarque :**

- Le code VLC proposé est un code préfixé et assure l'unicité de décodage.
- Le code proposé vérifie parfaitement l'inégalité de Kraft ($\sum_{i=1}^{N} 2^{-Li} \leq 1$)
- quel que soit la valeur de N, cela indique que le code est un code préfixé.

II. 1.4. Avantages du code 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.

- ✓ ***La flexibilité*** : le code est paramétrable. Plusieurs variétés du code peuvent être obtenues selon les paramètres : N, *preffix, suffix.*

II. 2. Les variétés des codes issus du code standard :

Le code standard proposé est de la forme C=*preffix.Suffix*. Ce dernier est très manipulable grâce à ca structure qui est constituée de deux parties, une fixe qui sert de champ de contrôle et une partie variables. Si par exemple nous avons la taille des mots à lire est égale à 8 donc N= 8, nous allons calculer la taille de la partie préfixe comme suit :

$$\text{Taille (preffix)} = \frac{\log(N)}{\log(2)}$$

Alors la partie préfixe est représentée sur 3 bits, puis on calcule la partie suffixe avec les informations fournies par la partie préfixe et aussi par l'indice à compresser.

Alors chaque traitement portant sur ces deux champs donne naissance à un nouveau code.

II. 2.1. Code adaptatif

Comme nous l'avons mentionné plus haut, la longueur du préfixe est donnée par $\log_2(N)$, tel que N représente le nombre de bits sur lequel sont codés les symboles, donc on aura 2^N symboles possibles a coder, et delà on peut déduire que la longueur de

Préfixe = $\log_2(\log_2(2^N))$.

Mais dans le cas ou le nombre de symbole est beaucoup inférieure a 2^N soit par exemple n ,alors la taille du préfixe sera $\log_2(\log_2(n))$ au lieu de $\log_2(N)$.

Or par cette stratégie on pourra minimiser de la taille du préfixe dans la plus part du temps.

✓ ***Exemple :***

Soit N=8 bits, le nombre de symboles (caractères) possibles à coder avec N est $2^N = 2^8 = 256$ Symboles. Et soit n=16 symboles dont nous avons besoin seulement de coder.
Longueur du préfixe :

- dans le 1er cas : préfixe = $\log_2(N) = \log_2(8) = 3$
- dans le 2ème cas : préfixe = $\log_2(\log_2(16)) = \log_2(4) = 2$

Donc la longueur du préfixe sera 2 au lieu de 3 et nous gagnerons ainsi 1 bit de la taille de préfixe.

II. 2.2. Code récursif

Un autre code peut être obtenu par le codage de la partie *suffix* lorsque celle-ci dépasse une certaine longueur. Dans ce cas c'est la partie *preffix* qui indique cette taille [L (*suffix*) = *preffix*], la partie *preffix* sert de champ de contrôle.

✓ **Codage**

Le code récursif suit les étapes suivantes :

- Choisir les valeurs de la partie préfixe (longueur de la partie suffixe) qu'on souhaite appliquer le code récursif.
- Appliquer le codage standard. Trouver $C_i=P_i.S_i$
- Si Pi ∈ [Valeurs choisies] Alors coder la partie suffixe S_i .trouver $S_i=P_j.S_j$. le code résultat est $C_i=P_i.P_j.S_j$

Sinon

- Le code résultat est : $C_i=P_i.S_i$

✓ **Décodage**

Le décodage suit les étapes suivantes :

- Lire la partie préfixe P_i.
- Si p_i ∈ [valeur choisies] Alors :

Lire P_j

Lire S_j (P_j bits)

Calculer $X=2^{pi}+2^{Pj}+S_j$

- Sinon

Calculer $X= 2^{pi} + S_i$

- **Optimisation de la compression**

Pour avoir un code VLC optimisé on doit respecter deux conditions

1. Le code satisfait la relation $\sum_{i=1}^{N} 2^{-Li} \leq 1$
2. Minimiser $\sum_{i=1}^{N} P(Si)\ L(Ci)$, cette valeur représente la moyenne de bits par symboles du code choisi.

III. Le processus de compression :

Du moment que le code proposé est flexible, on peut générer pour chaque ensembles de codes fixes qui sont représenter sur N bits, un code VLC correspondants. Cette propriété nous permet de considéré le fichier à compresser comme un vecteur de taille T (taille du fichier) de symboles, et chaque symbole est sur N bits.

Durant le processus de compression on cherche à trouver la meilleure valeur de N qui permet de minimiser au mieux $\sum_{i=1}^{N} P(Si)\ L(Ci)$.

- ***Deux stratégies peuvent être adoptées pour choisir la valeur N :***
  - **Statique :**

Cette stratégie dépend du type de fichier a compressé, on choisissant la valeur de N. Par exemple les fichiers de type texte on les considère comme un vecteurs de caractères ASCII,

donc N= 8bits, par contre les fichiers de types images peuvent êtres considérés comme un vecteur de pixels (3octet) donc N=24 bits et pour chaque type de données on aura son code VLC correspondants.

- **Dynamique :**

Cette stratégie préconise de choisir le N qui minimise le mieux la somme $\sum_{i=1}^{N} P(Si)\ L(Ci)$ Selon le processus suivant :

- Choisir une valeur pour N parmi l'ensemble E = {4, 8, 12, 16, 20, 28,32}
- Générer le code VLC pour N.
- Lire les symboles du fichier source (N bits pour chaque symbole)
- Calculer les P(Si)
- Calculer la somme $\sum_{i=1}^{N} P(Si)\ L(Ci)$.
- Comparer les différentes sommes et choisir le N qui donne la somme minimale.

IV. Objectif :

Dans notre applications nous voulons implémenter la méthode VLC de compression /décompression de données en utilisons le code proposé et cela en effectuant des codes préfixés ayant les plus petites longueurs possibles pour les symboles ayant les plus grandes probabilités, ce code doit impérativement satisfaire la condition de l'unicité de décodage, qui permet la décompression du fichier sans aucune ambigüité.

On va utiliser dans ce système une stratégie statique de compression sans perte de données, On utilisera une méthode de compression/décompression avec table basée sur une méthode de compression existante (compression standard).

Enfin on espères avoir de bon résultats de compression de cette méthode pour donner un nouvel élan a ce domaine qui a besoin de plus en plus de méthodes de compression efficaces.

V. Conception

V. 1. définition

La conception consiste à élaborer à partir de la spécification du problème une solution Informatique.

Pour la conception de notre application on a adopté une architecture flexible, basé sur une architecture modulaire.

- ✓ **Les modules** : sont des entités indépendantes intégrées dans une architecture pour produire une application.
- ✓ **Système** : l'ensemble des modules utilisés, ainsi que les relations qu'ils entretiennent entre eux.

V. 2. Les méthodes de conception :

Pour construire un système, il faut également une méthode de conception a suivre, pour cela on distingue deux familles de méthodes de conception :

V. 2.1. Les méthodes descendantes :

L'approche descendante commence par décomposer le problème initial en sous problèmes puis chaque sous-problème en de nouveaux sous-problèmes et ainsi de suite jusqu'aux problèmes que nous pouvons résoudre par des opérations primitives (ou des fonctions simples).

V. 2.2. Les méthodes ascendantes :

L'approche ascendante construit des opérations primitives que nous assemblons pour obtenir des opérations plus complexes et ainsi de suite jusqu'à une opération globale qui résout le problème initial.

On utilise les méthodes *descendante* dans le domaine du développement, parce qu'on en maîtrise en principe déjà les concepts dans le cadre d'une théorie qu'on met en œuvre, et au contraire les méthode ascendantes dans le cadre de la recherche, où on cherche à les faire émerger de la pratique.

V. 3. Cadre du travail :

Pour rejoindre l'objectif de notre application et réaliser toutes les taches essentielles, nous allons considérés la modularité comme un aspect fondamentale a la conception de l'application, et cela pour subvenir au besoin fixé en termes de simplicité des mécanismes, et faciliter le développement et la maintenance du système.

V. 4. Architecture générale de l'application :

Avant de représenter les différents modules de l'application, nous allons présenter l'architecture générale de la compression standard (une méthode déjà existante), puis l'architecture générale de la compression avec table qui est basé sur la compression standard.

- **La compression standard**

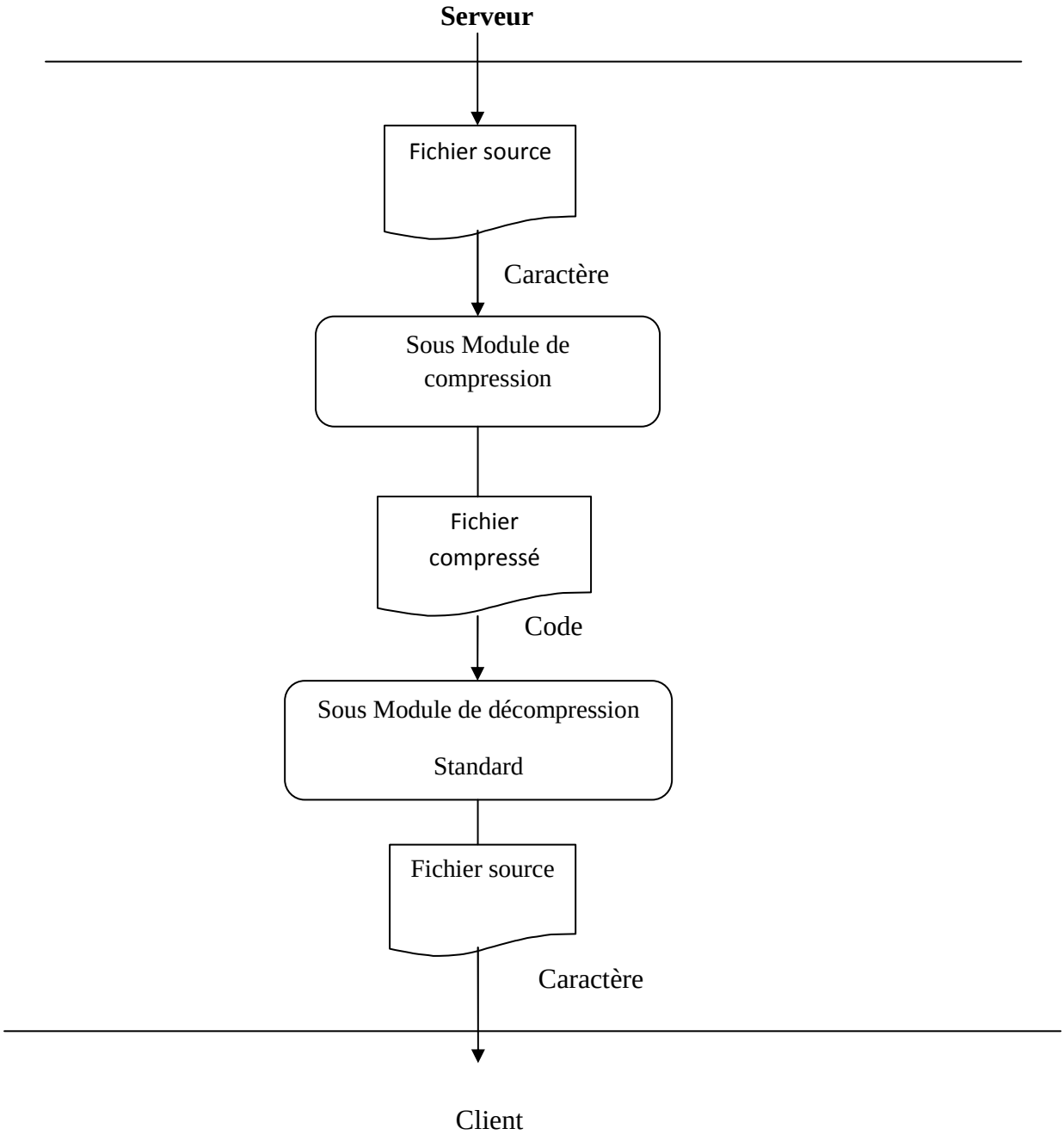


Figure 3.1: schéma de l'architecture générale de la compression standard

- **La compression avec table**

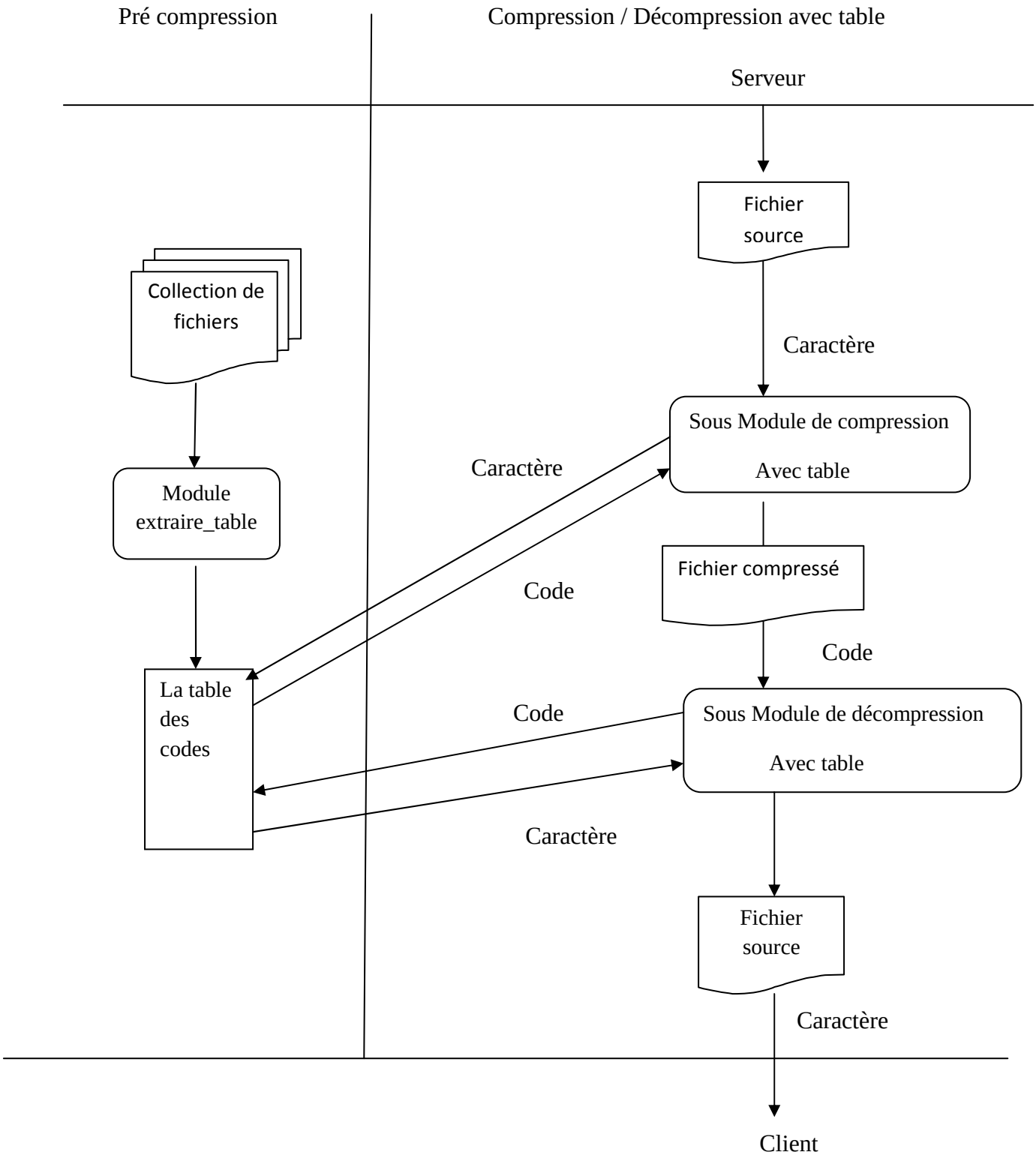


Figure 3.2 : schéma de l'architecture générale de la compression avec table

V. 5. Les Modules principaux de l'application (Le noyau) :

Avant de détailler les différents modules de notre application, il est nécessaire de donner les grandes lignes aux quelles ils feront référence, et cela afin d'atteindre notre objectif.

Nous allons ainsi utilisés les concepts vus au préalable pour définir les différents sous module qui la constituant.

Pour ce faire notre application doit offrir la possibilité de gérer les différentes taches, à savoir la compression/décompression standard, la compression/décompression avec table.
Donc nous pouvons décomposer le noyau de l'application en 3 modules :

1. Module extraire_table.
2. Module de compression/décompression avec table.
3. Module de compression/décompression standard.

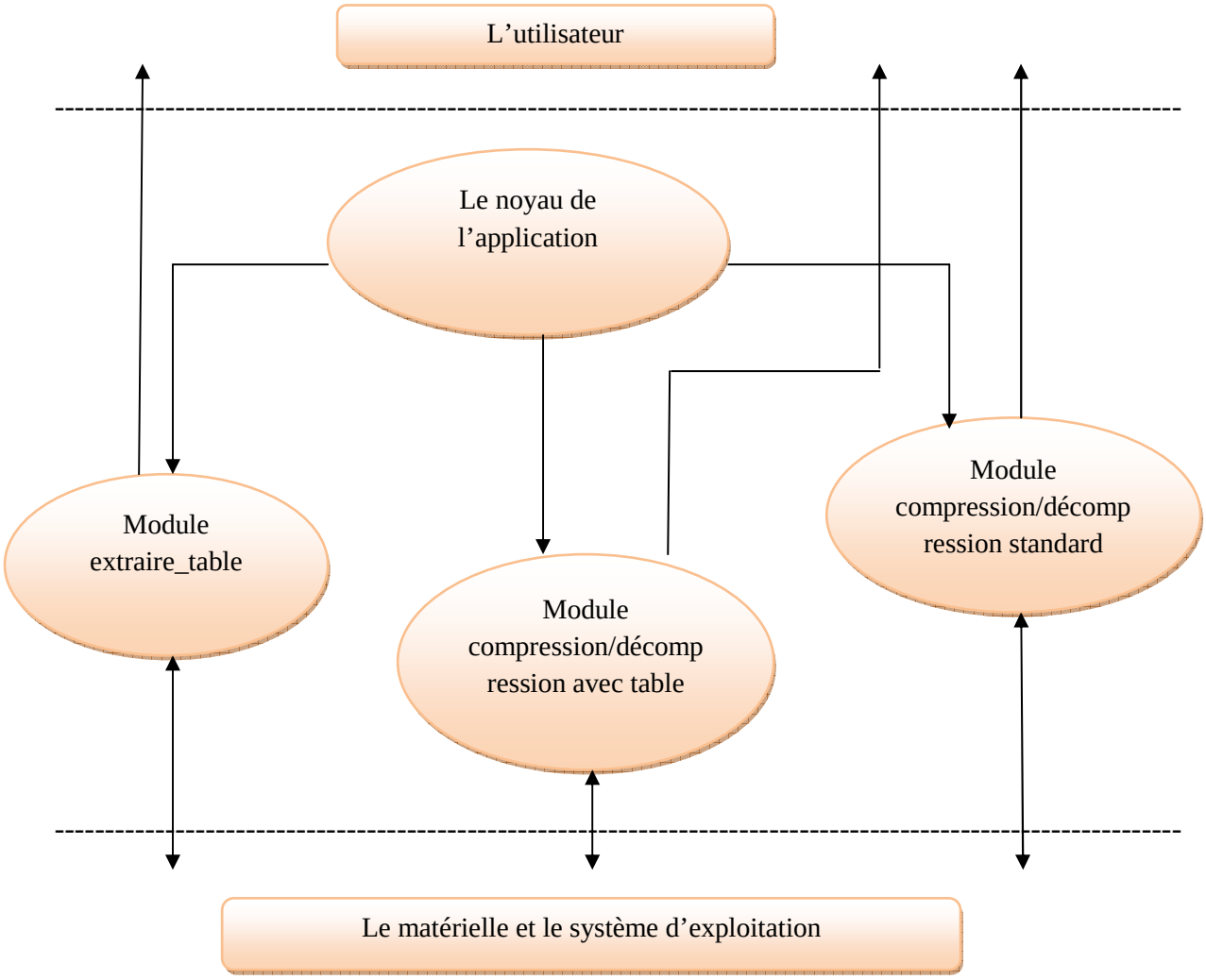


Figure 3.3: décomposition du module principale de l'application.

V. 5.1. Module extraire_table :

Le rôle du module extraire_table consiste à faire les taches suivantes :

- Lire chaque fichier et calculer le nombre d'occurrence de chaque symbole dans le fichier source, alors créer une table de fréquence qui prend les caractères comme indice de la table.
- trier et ordonner les fréquences calculées selon leur nombre d'occurrence, alors créer une table ordre de sorte que le caractère le plus fréquent on lui affecte le plus petit code.

- Créer une tables des codes ou au lieu de coder les symboles eux même, le processus de codage procède au codage de leur indice, ensuite il remplace chaque symbole par son nouveau code retourné.

Le module extraire_table est représenté dans la figure suivante :

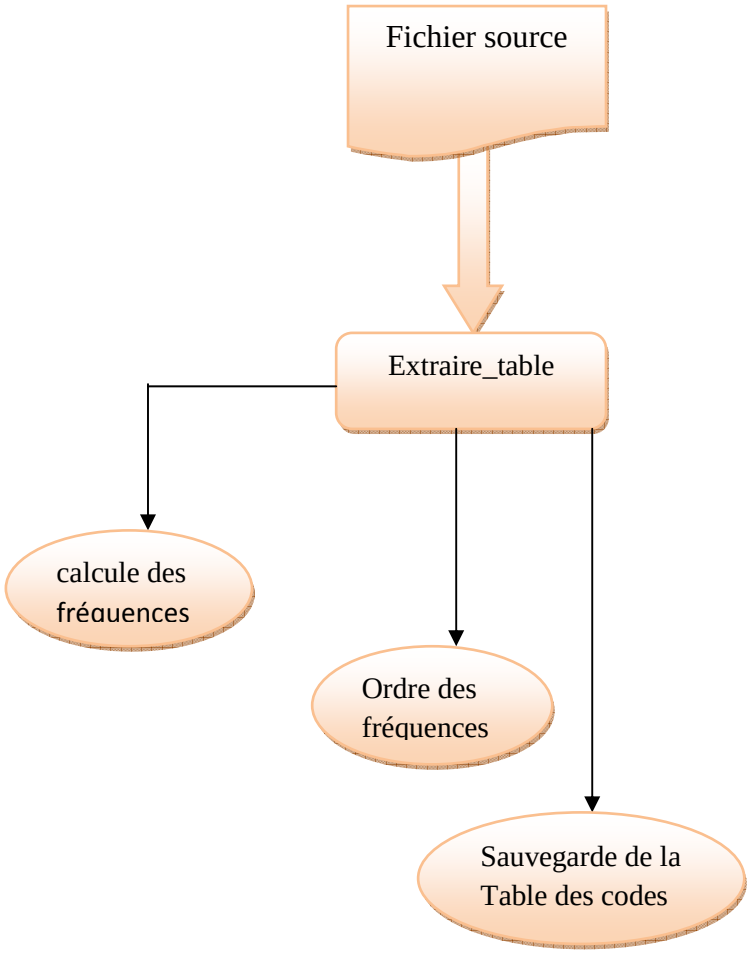


Figure 3.4 : schéma du module extraire_table

V. 5.2. Module de compression/décompression avec table :

Le module de compression/décompression avec table est décomposé en 2 sous modules :

- **Le sous module compression avec table :**

Lors du processus de compression, la fonction reçoit un flot de données caractère par caractère. Puis recherche son code associé on consultant la table des codes, ensuite calcule le

code VLC correspondant, qui renvoi comme résultat un code composé de deux parties : préfixe, suffixe.

Le sous module de compression est représenté dans la figure suivante :

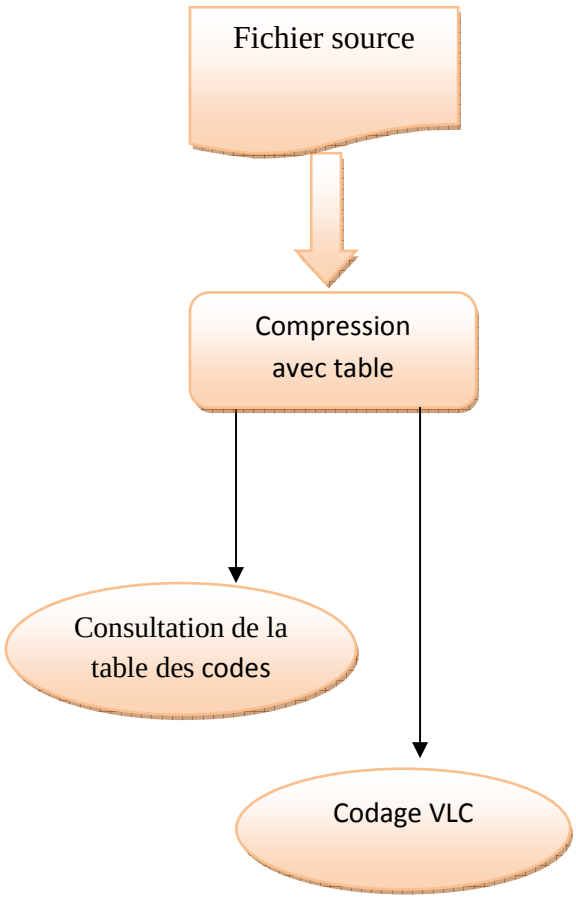


Figure 3.5 : schéma général de la compression avec table.

- **Le sous module de décompression avec table :**

Le principe utilisé pour la décompression permet de décompresser un fichier Compressé et de retrouver le même fichier source sans aucune perte de données. Pour cela la fonction de décompression commence à lire les information nécessaire au décodage des données (la taille du fichier source, le nombre de bits sur lequel sont codés les indices des symboles), puis reconnaitre les parties préfixe et la partie suffixe du morceau du code lu et cela pour passer ainsi au processus de décodage de ses parties, qui a pour but de retrouver l'indice initial codé. Enfin elle fait le travail inverse c'est-à-dire retrouver le symbole correspondant a l'indice

trouvé et cela en consultant la table des codes, et on retrouve alors le fichier source par la méthode de préfixe et suffixe.

Le sous module de décompression avec table est représenté dans la figure suivante :

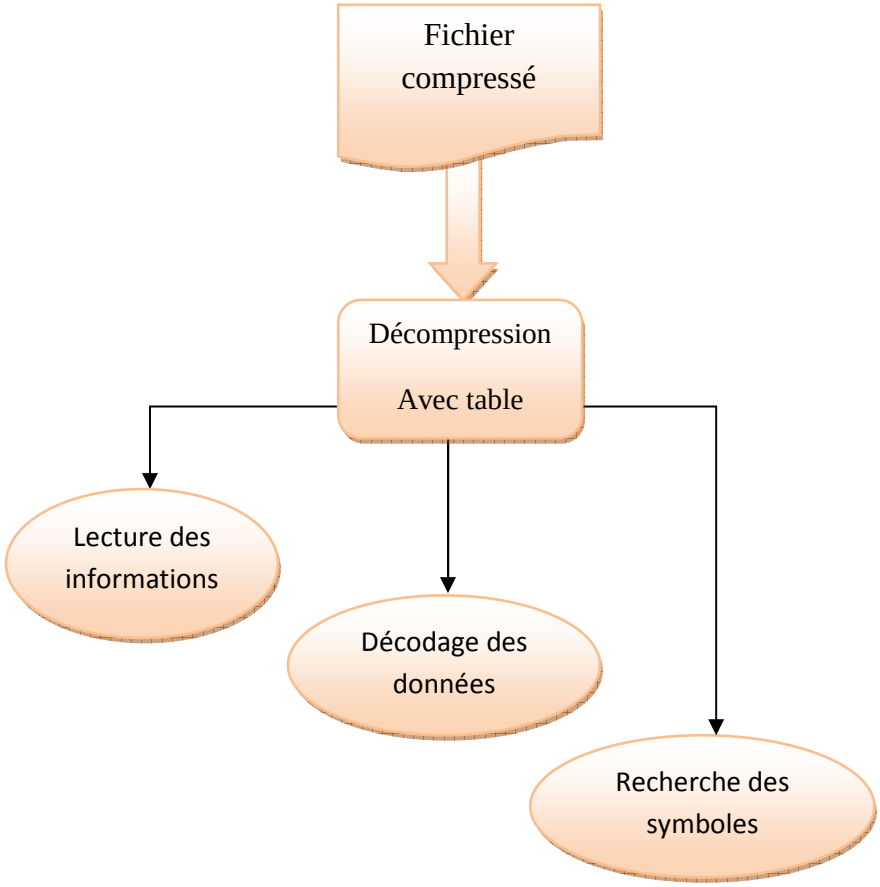


Figure 3.6 : schéma général de la décompression avec table.

V. 5.3. Module de compression/décompression standard :

Le module de compression/décompression standard est décomposé en 2 sous modules :

- **sous module de compression standard :**

Après avoir spécifié le fichier à compresser, le rôle principale de la fonction de compression standard consiste à calculer le nombre d'occurrences de chaque symbole dans le fichier source, les triées et les ordonner selon leurs nombres d'occurrences, qui a la fin retourne une table de tous les symboles triés et dont les indices prennent la place des

symboles et vis versa, ensuite elle se charge du processus du codage VLC. Enfin elle affecte et remplace chaque symbole par son nouveau code retourné.

Le sous module de compression standard est représenté dans la figure suivante :

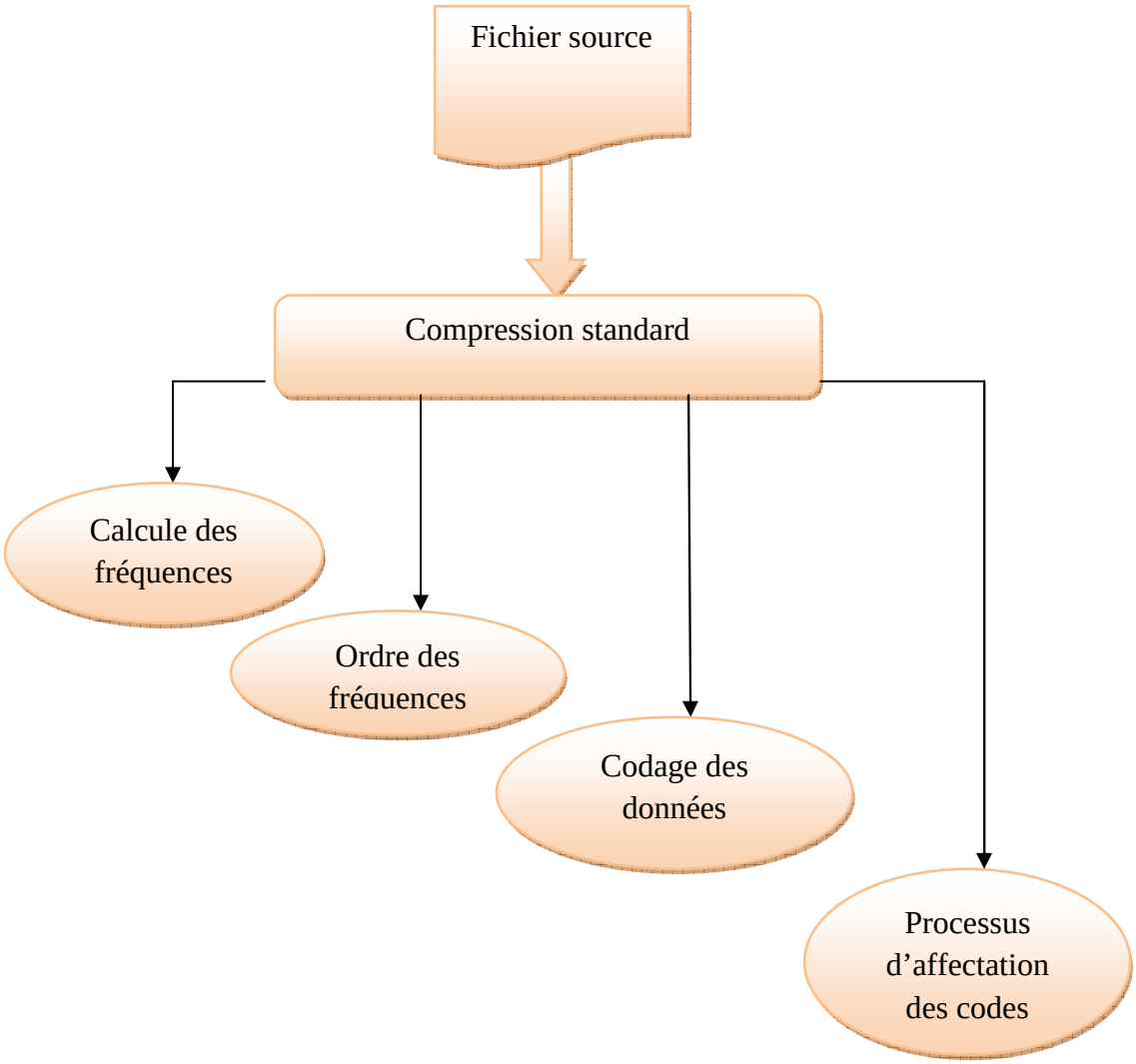


Figure 3.7 : schéma général de la compression standard

- **sous module de décompression standard :**

La décompression standard consiste a restituer les données initiales du fichier compressé, donc elle commence de lire les paramètres nécessaire au décodage des données, comme la taille du fichier source, le nombre de bits sur lequel sont codés les indices des symboles (N)…..etc. et qui sont inclus dans le fichier compressé.

Ensuite elle passe au processus de décodage pour générer l'ensemble des symboles de la chaine initiale.

Enfin elle fait le travaille inverse c'est-à-dire, retrouver le symbole correspondant a l'indice retourné et cela on consultant la table triée ajoutée au fichier compressé et on retrouve enfin le fichier source par la méthode de préfixe et suffixe.

Le sous module de décompression standard est représenté dans la figure suivante :

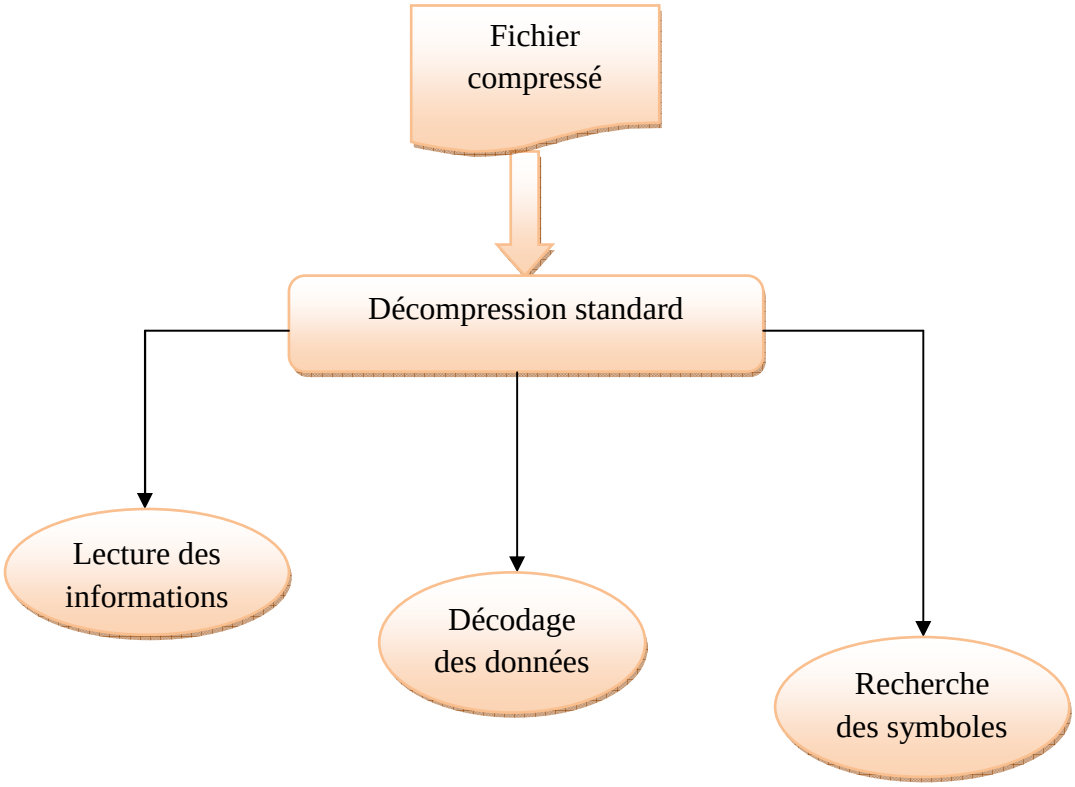


Figure 3.8 schéma général de la décompression standard

V. 6. Calcul et affichage des performances :

Grace au calcul et affichage des performances de l'algorithme, comme le temps et le taux de compression, que nous pouvons à la fin tester et évaluer cette méthode de compression.

V. 6.1. Le temps de compression

Pour calculer le temps de compression, ce processus utilise l'horloge système au début de la compression pour lui donner le temps de début de la compression. A la fin de la compression il sollicite une autre fois cette horloge pour lui donner le temps de fin de

compression. Et pour obtenir le temps exact de compression il fait la soustraction du temps fin de la compression par le temps début de la compression, comme suit :

Temps de compression = temps de fin – temps de début

V. 6.2. Le taux de compression

Pour calculer le taux de compression, le processus doit récupérer avant la compression la taille du fichier d'origine (source) et le fichier compressé (cible), puis il calcule le taux comme l'indique l'équation suivante :

$$\textbf{Taux de compression} = \frac{(taille\ originale - taille\ finale) * 100}{taille\ original}$$

VI. Les algorithmes :

On présente dans cette partie, les différentes algorithmes des modules essentiels qui forment le noyau de l'application, tel que la compression standard, la compression avec table, la décompression standard et la décompression avec table et d'autres modules qu'on a utilisés dans notre application.

VI. 1. Extraire la table des codes:

Debut

- nbr : le nombre de fichiers.
- Pour nbr fichiers faire
  Ouvrir le fichier
  Calculer la fréquence (nombre d'occurrence) de chaque symbole.
- Fin pour.
- Fermer les fichiers.
- *Trier et ordonner les symboles selon leurs nombres d'occurrences* (le plus

Fréquent, on lui affecte le plus petit code).

- Créer un nouveau fichier.
- Sauvegarder la table des codes.

Fin.

VI. 2 . Le Calcule du nombre d'occurrence de chaque symbole de N bits

Début

- Initialiser la table de fréquence a 0.
- Tant qu'il reste des bits dans le fichier faire

  Lire N bits.

  Freq[i]=Freq[i] +1.

- Fin tant que.

Fin.

VI. 3. Tri des symboles par leurs nombre d'occurrences :

Début

- Mettre les symboles ayant un nombre d'occurrence dans une table contenant le code ASCII du symbole et son nombre d'occurrence, soit Freq
- Fin=faux
- Initialiser la table ordre a (-1)
- Tant que vrai

  Indice = -1, max = 0, code = 0.

  - Pour i >=0

    Si freq[i] > max

    Indice=i

    Max=freq[i]

    Fin si.

  Fin pour.

  Si indice >= 0 alors

- Ordre [indice]=code

- Mettre le nombre d'occurrence de ce symbole dans la table Freq a 0

- Incrémenter code

Sinon

- Fin = vrai

Fin si.

Fin tant que

Fin.

VI. 4. La compression avec table

Debut

- Ouvrir le fichier a compressé.
- Créer un nouveau fichier (fichier de sortie).
- Charger la table des codes
- Ecrire l'entête dans le fichier de sortie.
- Tant que ce n'est pas la fin du fichier
  - Lire N bit du fichier source.
  - Trouver son indice (code ASCII) dans la table des codes.
  - Coder l'indice en utilisant la méthode de code préfix et suffixe.
  - Sauvegarder le code

Fin tant que

- Fermer le fichier

Fin.

VI. 5. La décompression avec table :

Début

- Ouvrir le fichier compressé.
- Créer un nouveau fichier.
- Charger l'entête du fichier compressé.

Tant qu'on n'a pas atteint la taille du fichier spécifié dans l'entête faire.

- Lire le préfixe.
- Lire le suffixe.
- Déduire l'indice du symbole.
- Trouver le code du symbole dans la table des codes.
- Sauvegarder le code du symbole dans le nouveau fichier.

Fin tant que

- Fermer le fichier.

Fin.

VI. 6. La compression par le code standard :

Début

- Ouvrir le fichier a compressé.
- Créer un nouveau fichier (fichier de sortie).
- Calculer le nombre d'occurrence de chaque symbole de N bits.
- Trier les symboles selon leurs nombre d'occurrences.
- Ecrire l'entête dans le fichier de sortie (N, la taille du fichier, le tableau des symboles)
- Tant que ce n'est pas la fin du fichier faire.
    - lire N bits du fichier source.
    - trouver l'ordre d'occurrence de ces N bits dans le fichier.

- coder l'ordre d'occurrence en utilisant la méthode du préfixe, suffixe.
- Sauvegarder le code dans le fichier créé.

- fin tant que

- fermer les fichiers.

Fin.

VI. 7. La décompression par le code standard :

Début

- Ouvrir le fichier compressé.
- Créer un nouveau fichier.
- Charger l'entête du fichier compressé.

Tant qu'on n'a pas atteint la taille du fichier spécifié dans l'entête faire.

- Lire le préfixe.

- Lire le suffixe.

- Déduire l'indice du symbole.

- Trouver le code du symbole dans le tableau des symboles.

- Sauvegarder le code du symbole dans le nouveau fichier.

Fin tant que

- Fermer le fichier.

Fin.

VI. 8. Le calcul de la taille d'un code :

Début

-calculer le nombre de symboles n.

- calculer le nombre de bits T nécessaires pour coder un symbole ($T = \log_2(n)$).

- P = position du premier bit significatif dans l'indice associé au symbole.

- Si l'indice de symbole = 0 ou 1 alors

  - Retourner T+1.

Sinon

  - Retourner T+P-1.

Fin si.

Fin.

VI. 9. Le calcule de la taille d'une compression

Début

- Ouvrir le fichier.
- Calculer les nombres d'occurrence des symboles de N bits.
- Pour chaque symbole existant calculer sa taille compressée.
- Calculer l'espace occupé par un symbole (espace =taille_symbole * nombre d'occurrence).
- Calculer la somme des espaces occupés par tous les symboles.
- Ajouter la taille de l'entête pour trouver la taille totale.

Fin.

VI. 10. Codage d'un symbole par le code préfixe et suffixe :

Début

$n=\log_2$ (nombre de symboles).

1. i=indice du symbole dans la table ordonnée.
2. Décalage a gauche de i, (taille de i-Taille de code de symbole) fois.
3. Poser t=taille de code de symbole.
4. Décalage a gauche de i par un bit.
5. Si le bit sortant est à 1 alors aller a 9
6. Décrémenter t.
7. Si t=0 alors aller a 14

8. Aller a 4.
9. Décrémenter t.
10. Préfixe = t.
11. Si t != 0 alors aller a 13
12. i=1.
13. Suffixe = i. aller a 16.
14. Suffixe= 0.
15. Préfixe = 0.
16. Sauvegarder préfixe sur $\log_2(n)$ bits.
17. Sauvegarder suffixe.

Fin

VI. 11. Décodage de l'indice d'un symbole codé par le code préfixe et suffixe

Etant donné qu'on a le préfixe et le suffixe.

Début

1. Décalage a gauche du suffixe (taille de la variable de suffixe-préfixe)fois.
2. i=1.
3. Si préfixe=0 alors aller a 10.
4. Décalage a gauche de i par un bit.
5. Décalage a gauche du suffixe par 1 bit.
6. i =i+bit sortant du suffixe.
7. Décrémenter le préfixe.
8. Si préfixe>0 alors aller a 4.
9. Aller a 12.
10. Si suffixe !=0 alors aller a 12.
11. i = 0.
12. Retourner i.

Fin.

VII. conclusion

Dans ce chapitre nous avons présenté la partie d'analyse et de conception d'un système de compression / décompression, cette étape est essentielle car elle nous aide à organiser notre application ou on a adopté une approche modulaire ce qui nous a conduit a subdivisé le noyau en modules, et chaque module en sous modules, puis nous avons spécifié les différents algorithmes qui composent notre système.

Dans le chapitre qui suit nous allons faire l'implémentation de l'application en se basant sur cette conception.

Chapitre 4 :

Implémentation et évaluation

I. Introduction

Dans ce chapitre nous allons présenter en premier lieu l'environnement de développement de notre application, ainsi le langage de programmation utilisé, en donnant les raisons pour les quelles on a choisi le langage de programmation c++,puis on va présenter les fonctions principales du noyau, ensuite on va évaluer les deux méthodes de compression standard et avec table.

Les tests de ces deux méthodes sont présentés sous forme des diagrammes formés par des axes

- Axe taux de compression : ces tests ont pour objectif d'évaluer les performances des deux méthodes (standard et avec table) de point de vue taux de compression.
- Axe temps de compression : ces tests ont pour objectif d'évaluer les performances des deux méthodes de point de vue temps de compression.

II. Environnement de développement

Nous avons choisi pour la réalisation de notre travail le système d'exploitationMicrosoft Windows7. Ce choix est fait car les systèmes d'exploitation Windowssont très utilisés grâce à leurs simplicités, efficacités, convivialités et fiabilités.

L'ordinateur sur lequel nous avons fait ces évaluations est doté d'un processeur de fréquence 2.67 GHz et d'une mémoire vive de 4GO.

Pour ce qui concerne la programmation, nous avons choisi le langage c++ et Dev-c++
Comme environnement de développement de l'application.

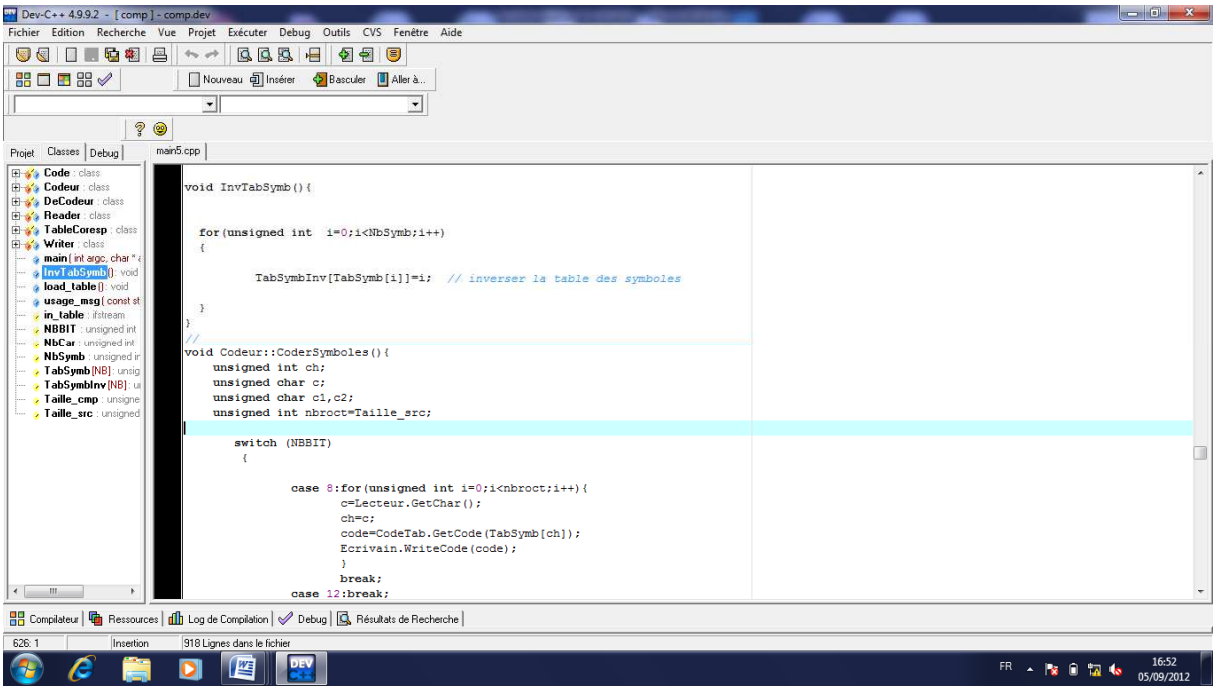


Figure 4.1 interface de l'environnement de développement Dev c++.

II. 1. Description du langage c++

II. 1.1 Historique [20] :

Le langage C++ a été développé par« Bjarne Stroustrup » au cours des années 1980, alors qu'il travaillait dans le laboratoire de recherche Bell d'AT&T. Il s'agissait en l'occurrence d'améliorer le langage C. Il l'avait d'ailleurs nommé *C with classes*(C avec des classes). Les premières améliorations se concrétisèrent donc par la prise en charge des classes, ainsi que par de nombreuses autres fonctionnalités, En 1989, c'est la sortie de la version 2.0 de C++. Parmi les nouvelles fonctionnalités, il y avait l'héritage multiple, les classes abstraites, les fonctions membres statiques, les fonctions membres constantes, etc….

Comme le langage C++ évoluait, la bibliothèque standard évoluait de concert. La première addition à la bibliothèque standard de C++ concernait les flux d'entrées/sorties qui apportaient les fonctionnalités nécessaires au remplacement des fonctions C traditionnelles telles que *printf* et *scanf*. Ensuite, parmi les additions les plus importantes, il y avait la Standard Template Library.

En langage C,++ est l'opérateur d'incrémentation, c'est-à-dire l'augmentation de la valeur d'une variable de 1. C'est pourquoi C++ porte ce nom : cela signifie que C++ est un niveau au-dessus du C.

II. 1.2 Pourquoiutiliser le C++

- ✓ Il existe de nombreuses bibliothèques C++ en plus de la bibliothèque standard de C++ (*C++ Standard Library*). Par ailleurs, C++ permet l'utilisation de l'ensemble des bibliothèques C existantes.
- ✓ Le c++ est un langage permettant la programmation sous de multiples paradigmes comme la programmation orientée objet qui est très importante dans le développement des applications.
- ✓ Les compilateurs c++ sont actuellementimplémentés 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 rapide.

- ✓ La plupart des implémentations des algorithmes standards sont implémentés à base de langage c++. Il est actuellement le 3^{e} langage le plus utilisé au monde.

III. Les fonctions principales du noyau :

Nous allons présentés dans cette partie les principales fonctions utilisées par notre application, on présentant ainsi le rôle de chacune, et cela par apport aux deux méthodes de compression utilisées, standard et avec table.

III. 1. La Compression standard :

voidSetCode(unsignedint val): cettefonction permet de calculer la valeur de préfixe et sufixe ,ainsi de calculer la taille de suffixe et la taille de code.

voidCalculFreq () : cette fonction permet d'initialiser le vecteur de fréquences, puislire le Fichier octet par octet pour calculer les fréquences.

voidTri_TabFreq() : cette fonction trie la table des fréquences afin d'avoir une table de symboles ensuite ,ensuite elle inverse la table des symboles, c'est-à-dire

parcourir la table triée et mettre l'indice de la case dans la case qui correspond au contenu de cette case..

voidHeader_write() : cette fonction permet d'écrire le nombre de symboles, le nombre taille symbole, la Taille du Fichier et le Tableau des symboles dans le fichier de sortie .

voidCoderSymboles() : après l'inversement de la table des symboles ,cette fonction envoie la chaine du Code dans le fichier de sortie.

voidencoder(string ifile, string ofile, boolverbose, char type) : cette fonction s'occupe de la lecture du fichier source et la création du fichier final, fait les appels à la fonction de codage est écris le résultat dans le fichier final.

voiddecoder(string ifile, string ofile, boolverbose, char type) : le rôle de cette fonction est comme son nom l'indique, elle parcourt le fichier compressé, lit l'en-tête qui l'accompagne puis lire le préfixe et le sufixe pour décoder l'indice de symbole et le décompresse vers un nouveaux fichier de sortie.

voidFindPreffixSize(intnbits) :cette fonction permet de trouver la taille de préfixe

III. 2. La Compression avec table

int lire () : le rôle de cette fonction est de parcourir tous les fichiers existant dans le dossier « data » et lire octet par octet, puis calculer les fréquences afin de remplir la table des fréquences.

voidtab_ord() : après le calcule des fréquences cette fonction permet d'ordonner la table des fréquences pour construire la table des codes.

voidsav_tab(int t[],long int n) : après l'ouverture d'un nouveau fichier , cette fonction permet de sauvegarder la table des codes.

voidload_table() : cette fonction permet d'ouvrir un fichier en lecture et de charger la table des codes.

void Codeur::Coder(): cette fonction appel en premier la fonction de chargement de la table, ensuite il initialise la table des codes ,ainsi l'écriture de l'entête dans le fichier de sortie et fait appel a la fonction de codage.

IV. Evaluation

IV. 1. Evaluation du taux de compression

Pour évaluer le taux de compression nous allons utiliser les résultats donnés par l'application, ces derniers sont calculés par une fonction spéciale.

L'équation pour calculerle taux de compression sous forme de pourcentage est la suivante :

$$\textbf{Taux de compression} = \frac{(tailleoriginale - taillefinale) * 100}{tailleoriginal}$$

Nous devons faire cette évaluation et comparaison entre la compression standard et la compression avec table par rapport à deux critères : le type de Fichier (texte,htm,image) et aussi à la longueur du mot à lire (N).

- **Les fichiers textes**

On prend un ensemble de fichiers textes, a taille moyenne de ces fichier est de 700 k octets et on prend la longueur du mot à lire (N) sur 8 bits, puis sur 16 bits.

Les résultats obtenus sont représentés sur les figures suivantes :

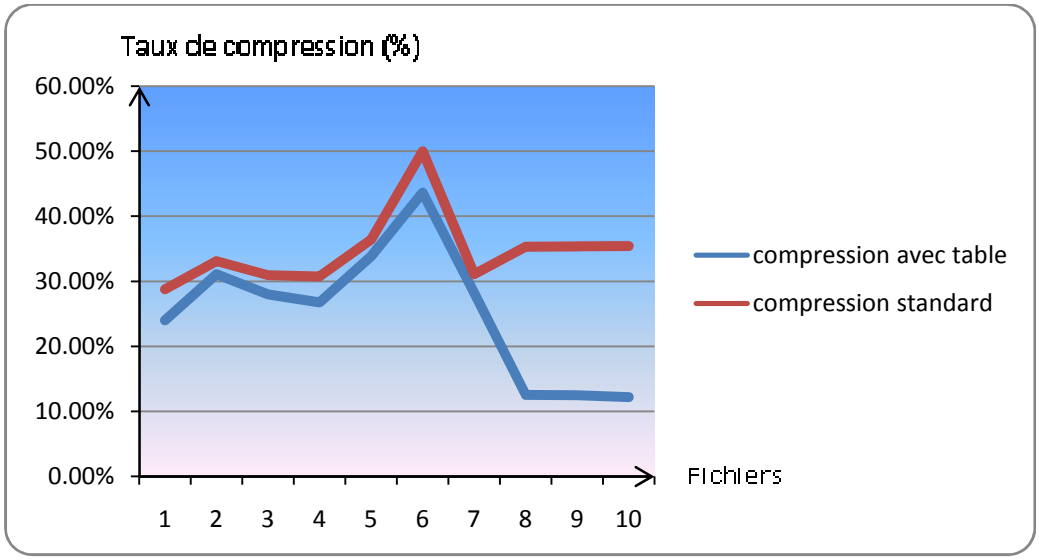


Figure 4.2 Evaluation du taux de compression pour les fichiers textes sur 8 bits.

Nous remarquons à partir de la figure précédente que pour la valeur N = 8 letaux de compression de la méthode standard est un peu mieux que celle de la méthode de compression avec table et ceci pour tous les fichiers textes choisis.

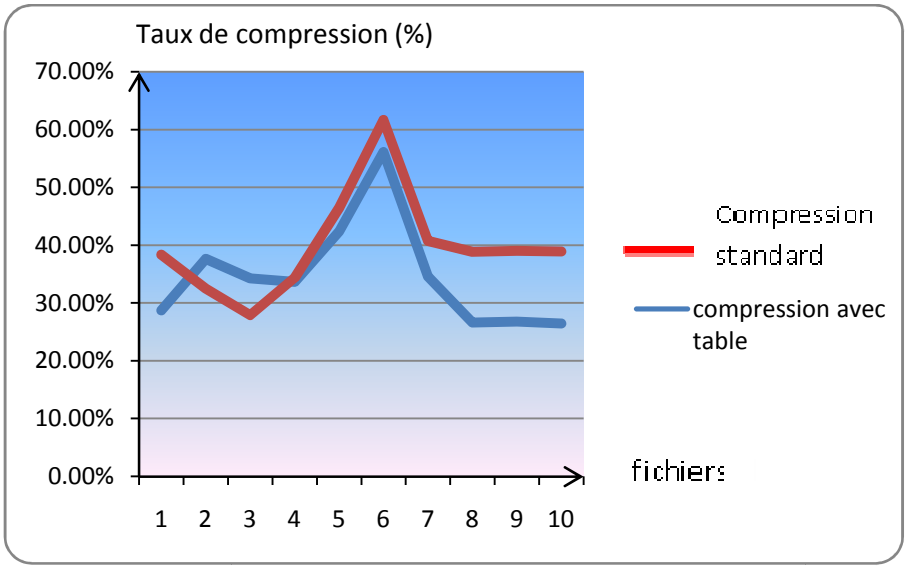


Figure 4.3 Evaluation du taux de compression pour les fichiers textes sur 16 bits.

Nous remarquons à partir de la figure précédente que pour la valeur de N= 16 le taux de compression de la méthode avec table est un peu mieux dans certains fichiers que celui de la méthode standard.

- **Les fichiers htm**

On prend un ensemble fichiers htm(pages web), a taille moyenne de ces fichier est de 200koctets et on prend la longueur du mot a lire (N) sur 8 bits, puis sur 16 bits.
Les résultats obtenus sont représentés sur les figures suivantes :

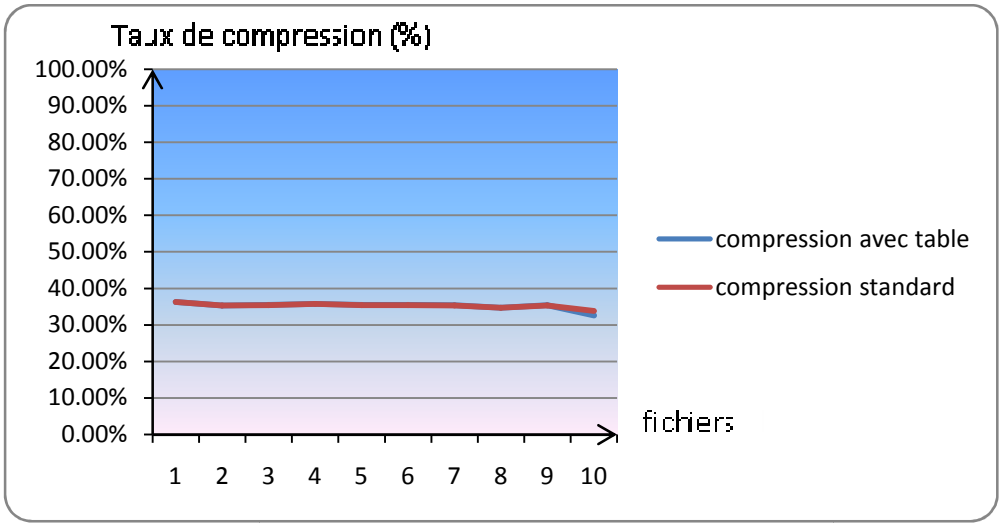


Figure 4.4 Evaluation du taux de compression pour les fichiers htm sur 8 bits.

Nous remarquons à partir de la figure précédente que pour la valeur N =8 le taux de compression avec les deux méthodes standard et avec table, est presque le même pour tous les fichiers textes choisis.

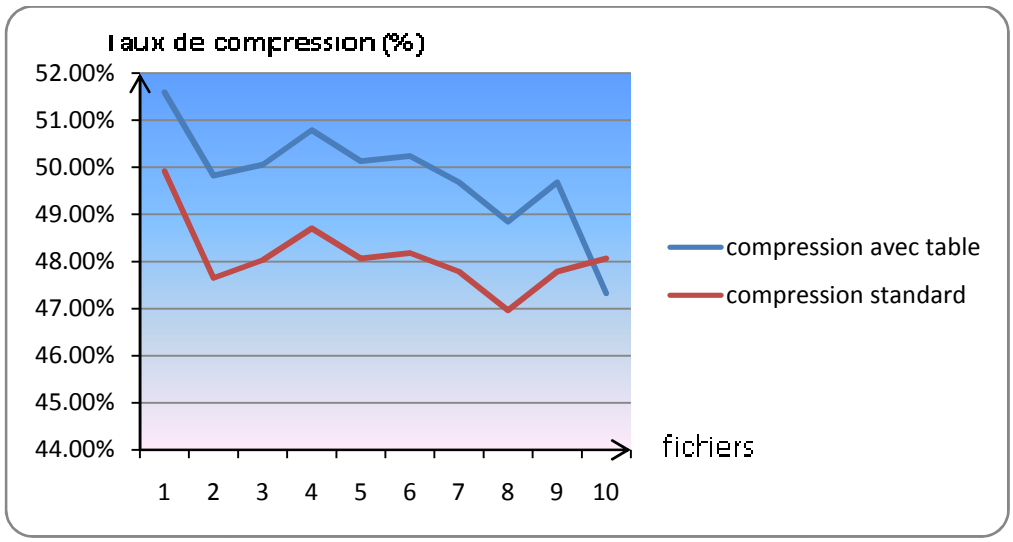


Figure 4.5 Evaluation du taux de compression pour les fichiers htm sur 16 bits.

Nous remarquons à partir de la figure précédente que pour la valeur de N= 16 le taux de compression de la méthode avec table est beaucoup mieux que la méthode de compression standard dans la plupart des fichiers choisis.

- **Les fichiers images**

Pour l'évaluation du taux de compression des images nous avons pris une collection d'image de format tiff, la taille moyenne de ces images est de 5M octets et on prend N sur 8 bits, puis sur 16 bits.
Les résultats obtenus sont représentés sur les figures suivantes :

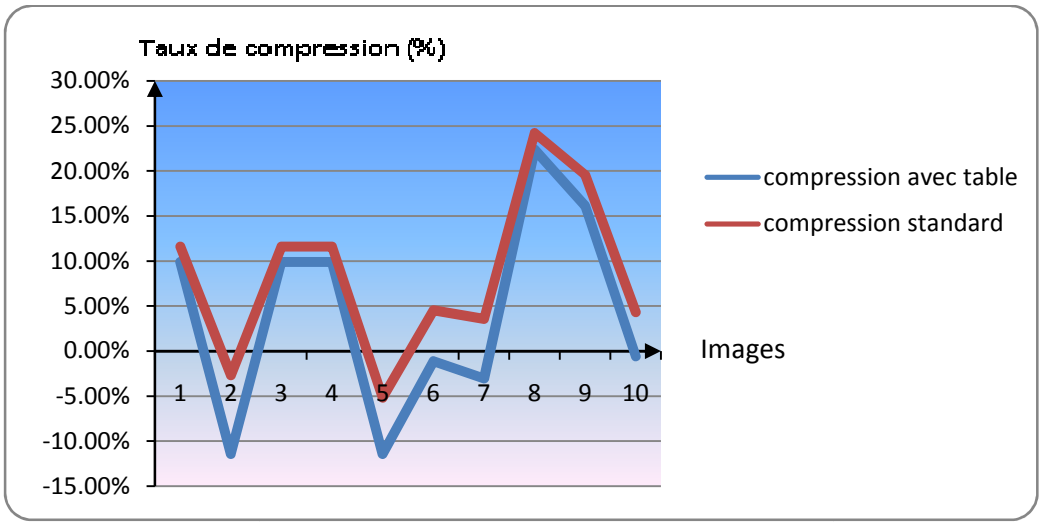


Figure 4.6 Evaluation du taux de compression pour les images sur 8 bits.

Nous remarquons dans cette figure le taux de compression avec la méthode standard est un peu mieux que celle de la méthode de compression avec table pour N=8 et ceci pour tous les fichiers images choisis.

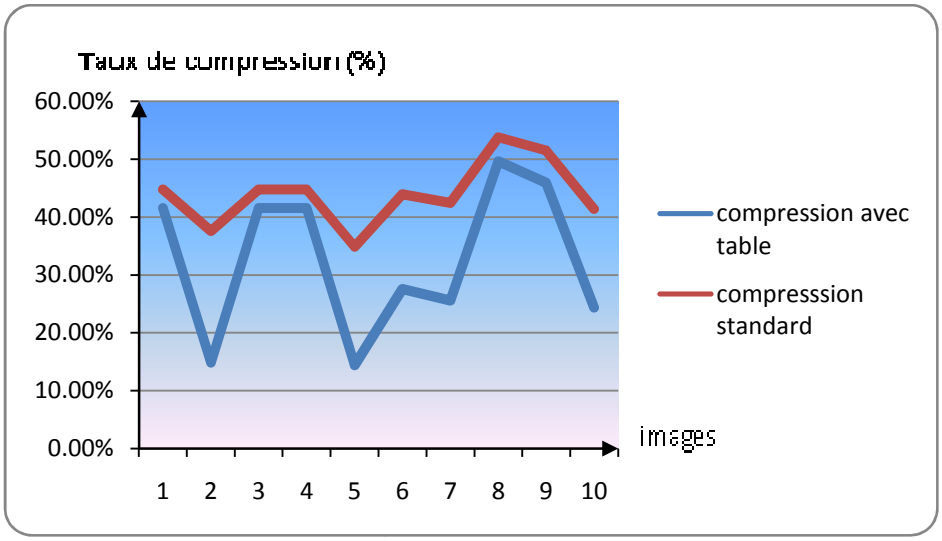


Figure 4.7 Evaluation du taux de compression pour les images sur 16 bits.

Nous remarquons dans cette figure que le taux de compression avec la méthode standard est un peu mieux que celle de la méthode de compression avec table pour N=16 et ceci pour tous les fichiers images choisis.

- **Interprétation des résultats de taux de compression**

✓ Nous remarquons dans les figures précédentes que le taux de compression avec N=16 est beaucoup mieux que celui de N=8 et celui-ci pour tous les types de fichiers (texte, htm, image) et pour les deux méthodes de compression, car avec la valeur N =16, on peut avoir une table des codes de taille de 2^{16} ,c'est-à-dire avoir une table de code plus grande, afin de coder tous les symboles contenus dans les fichiers.

✓ Pour les fichiers images on ne peut pas justifier exactement ces résultats vue que l'unité de codage chez les images est sur 24 bits.

✓ Pour les fichiers texte et htm, la méthode standard est meilleur dans certains fichiers, car avec cette méthode de compression, chaque fichier a son propre table de code qui est nécessaire pour la compression, par contre dans la compression avec table on va extraire la table des codes une seule fois pour tous les fichiers ,ainsiles fréquences d'apparitions de symboles sont différentes des fréquences des symboles de la table des

codes,et cela peut donner un taux de compression moins par apport a la méthode standard.

Pour avoir les fréquences exactes des symboles, il faut choisir une collection de fichier plus grandes(exemple un corpus de 1000 fichiers demême type) et cela donne un taux de compression meilleur pour la méthode avec table.

IV. 2. Evaluation du temps de compression :

- **Le calcule de temps de compression**

le temps de compression est calculé dans le module de compression/décompression et cela par l'appelle de la fonction horloge( ) avant le début de la compression ,pour donner le temps de début de la compression, a la fin de la compression il appelle une autre fois la fonction horloge( ) ,pour donner le temps de la fin de la compression , a la fin pour obtenir le temps de compression exacte, il fait la soustraction du temps fin de la compression par le temps début de la compression.

- **Le calcule de temps de décompression**

Pour calculer le temps de décompression on utilise l'horloge système dans le module de compression/décompression et cela par l'appelle de la fonction horloge( ) avant le début de la décompression ,pour lui donner le temps de début de la décompression, a la fin de la décompression il appelle une autre fois la fonction horloge( ) ,pour lui donner le temps de la fin de la décompression , a la fin pour obtenir le temps de décompression exacte, il fait la soustraction du temps fin de la décompression par le temps début de la décompression.

Dans cette partie d'évaluation, nous allons comparer les deux types de compression (standard et avec table) par rapport à leur temps d'exécution. Les valeurs du temps représentées dans les figures qui suivent, sont ceux calculées à partir des tests précédents pour calculer les taux de compression. Donc Nous avons calculé le temps de compression pour chaque type de fichier et pour chaque valeur de N. Le temps est représenté en seconde.

- **Les fichiers textes**

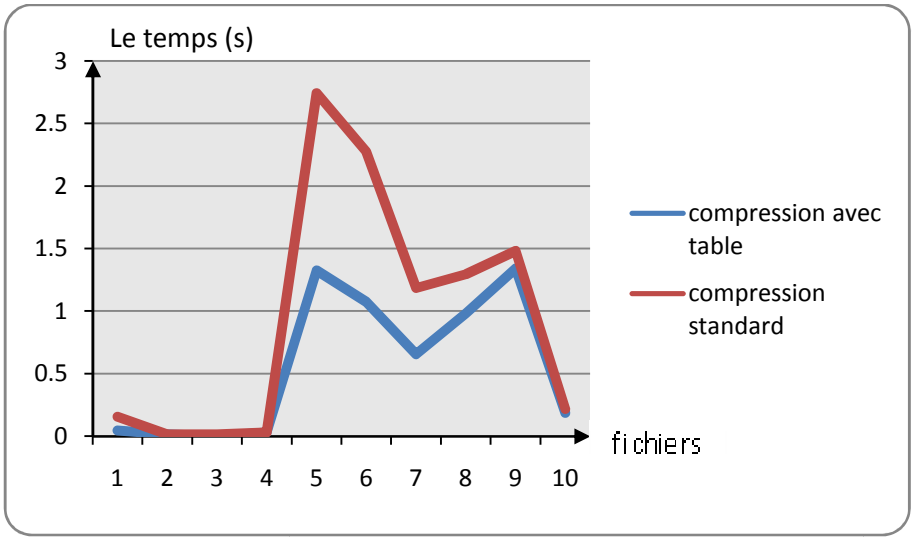


Figure 4.8 Variation du temps de compression des fichiers textes sur 8 bits.

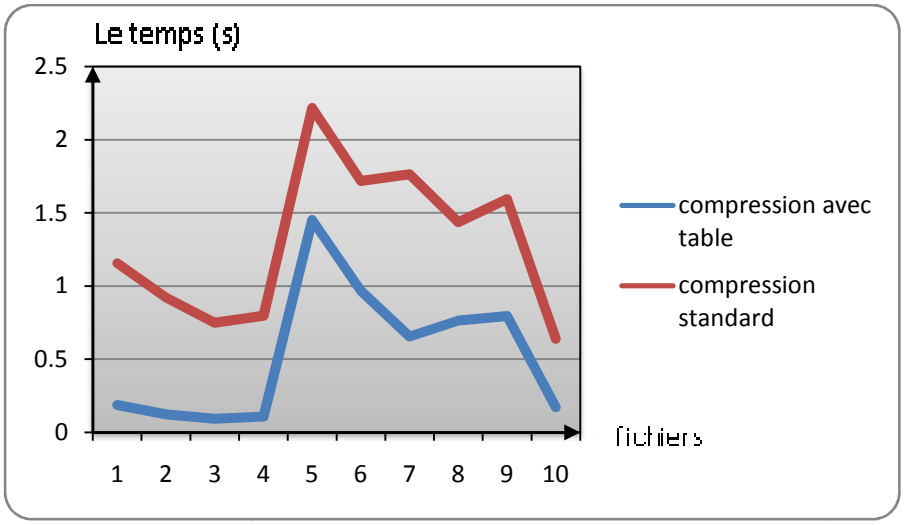


Figure 4.9 Variation du temps de compression des fichiers textes sur 16 bits.

- **Les fichiers htm**

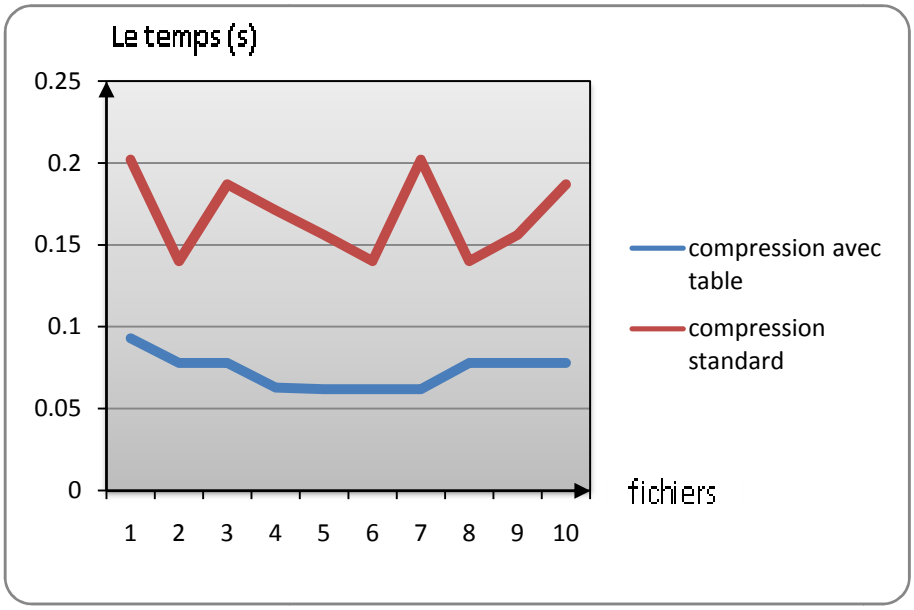


Figure 4.10 Variation du temps de compression des fichiers htm sur 8 bits.

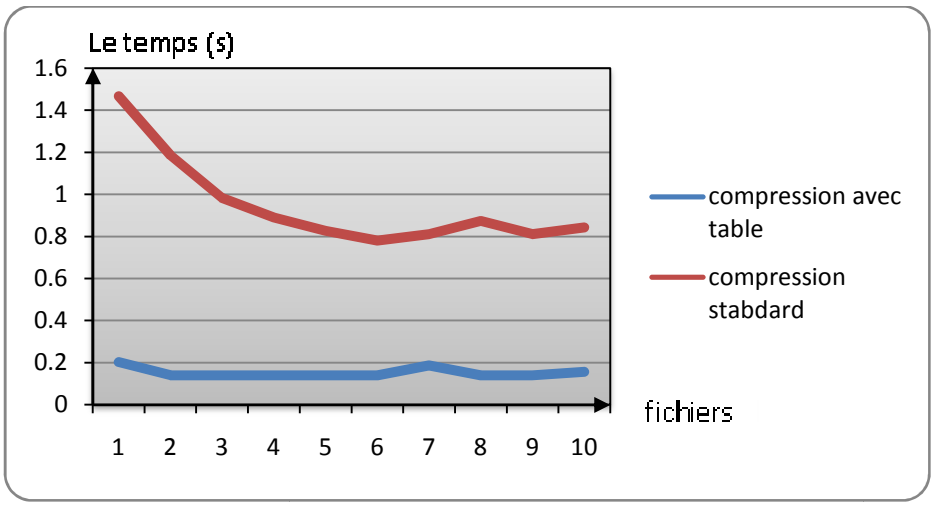


Figure 4.11 Variation du temps de compression des fichiers htm sur 16 bits.

- **Les images**

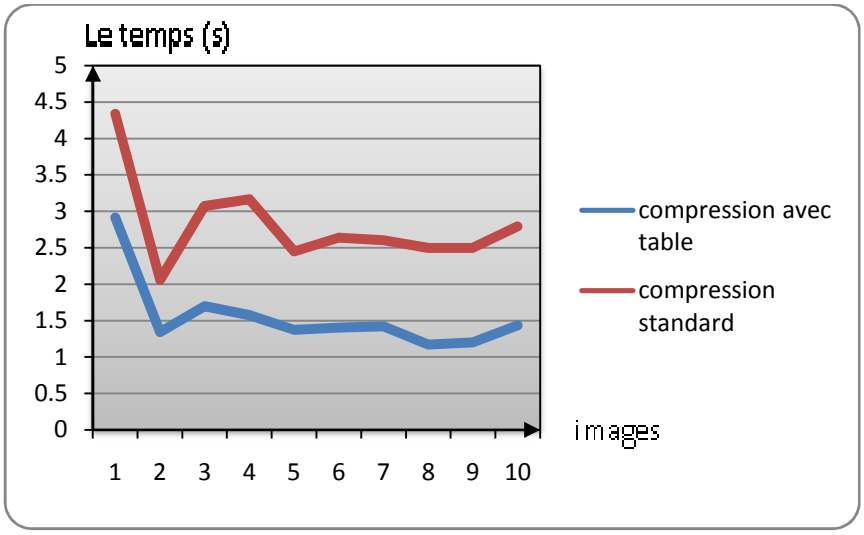


Figure 4.12 Variation du temps de compression des images sur 8 bits.

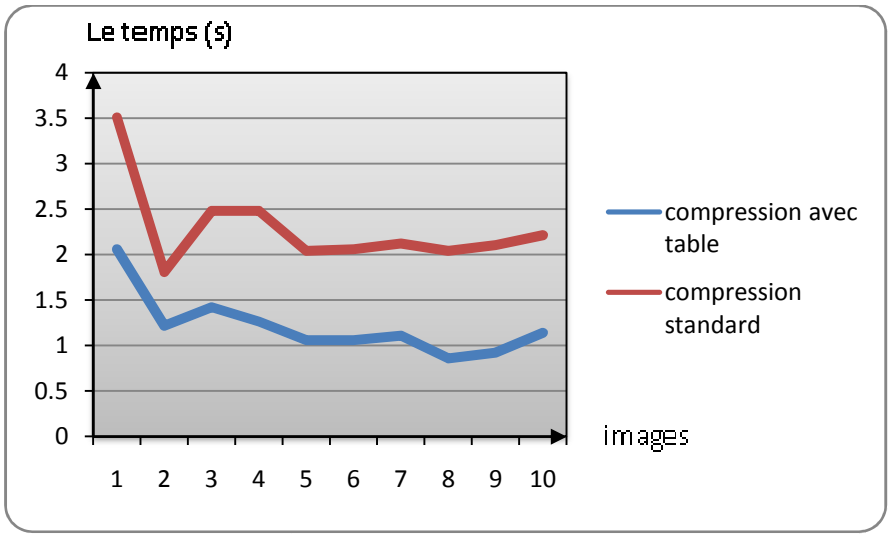


Figure 4.13 Variation du temps de compression des images sur 16 bits.

- **Interprétation des figures de l'évaluation du temps de compression :**

Nous remarquons à partir des 6 figures précédentes que le temps de compression esttoujours meilleur pour la compression avec table quelque soit le type du fichier ou bien la valeur deN.

La différence est plus grande avec la valeur N=16 bits que de 8bits et cela pour tous les types de fichier. ainsi,le temps de compression est petit avec la valeur N=8 et grand pour N=16, en effet, le temps de compression augmente avec N , car la taille maximale de la table est 2^N ,et le temps de tri de cette table dépend évidemment de ca taille.

Nous remarquons que le temps d'exécution de la compression par le code standard est toujours plus grand par apport au code avec table et cela dans tous les cas, car le processus de compression avec table et beaucoup plus simple et moins complexe que celui de la compression standard a cause des calculs qu'il faut faire pour le codage des données dans chaque fichier.

Ce qui concerne le temps de décompression on peut dire qu'il est linéaire avec la taille du fichier.et il n'y a pas une dépendance remarquable ni avec le type de fichier ni avec la taille de bloc N.

V. Conclusion :

Dans ce chapitre on a vue l'environnement de travail utilisé, ainsi que les différentes fonctions qui constituent notre application, ensuite on a effectué des tests pour évaluer le taux et le temps de compression,et cela pour les deux méthodes de compression standard, et avec table, donc on a présenté les différentes résultats obtenus après ces tests suivis de commentaires et de justifications.

Les résultats obtenus montrent quele taux de compression pour la méthode de compression avec table est mieux par apport à la méthode standard dans certains fichiers surtout pour les pages web et cela dépend de la table des codes.

Les résultats montrent aussi quele taux de compression pour les deux méthodes, ne dépend pas uniquement de la taille des fichiers mais d'une autre variante qui est la taille de la table de code. Si la taille de la table est grande on aura un meilleur taux de compression, c'est pour cela il est nécessaire de choisir des tailles (N) qui donnent des tables assez grandes.

Conclusion générale

Ce modeste travaille nous a permis, de découvrir le monde de la compression de données en passant en revue les différentes méthodes et techniques de compression de données, il nous a données l'idée sur les différences entre ces méthodes telle que chaque algorithmes possède ces propre caractéristiques.

Notre travail avait comme objectif l'implémentation et l'évaluation d'une nouvelle méthode de compression de données sans perte, basé sur le code VLC, pour ce faire il est nécessaire de comprendre divers concepts dans le monde de compression de données et de la programmation

A la fin de notre travail, nous avons constaté que cette nouvelle méthode de compression de données donne de meilleurs résultats de taux de compression pour les fichiers web, et cela dépond de la table des codes, et pour un meilleur taux de compression avec table, il faut choisir une collection de fichiers assez grande afin d'obtenir les fréquences exactes des symboles.

Cette expérience nous a permis donc d'acquérir beaucoup de connaissance dans un monde aussi vaste qui est la compression de données, les différents algorithmes utilisés ainsi de mieux se familiariser avec le langage C ++.

Nous précisons que les différentes parties de notre application peuvent être maintenus et optimiser pour donner encore de bons et de meilleurs résultats.

Référence

[1]: Sciences de l'information et de la communication, Bougnoux Daniel Édition Larousse, collection Textes essentiels, 1994.

[2] : Transmission de l'information, cours de l'EPU de Tours.

[3] : Thèse de Guillaume BOISSON Représentations hautement scalables pour la Compression vidéo sur une large gamme de débits/résolutions, 2005

[4] : Pascale Plume .compression de données .méthodes ,algorithmes ,programmes détailles .EDITION EYROLLES.

[5]:Mémoire de magister dispositif hard ware de compression de données pour cache disc multi port série « HAMEG SAMIR » 2002.

[6]: Introduction to data compression, Khalid Sayood, 2006, Morgan Kaufmann.

[7]: les formats de compression d'image, MICHOT JULIEN, 2004.

[8] :Sebastian THON License Pro Sil IN 2010-2011.

[9]: T.I.P.E la compression JPEG, FLIEDEL ROMAIN, 2005.

[10]: MPEG-2, John Watkinson, Focal Press, 1999.

[11]: data compression, the complete reference, David Salomon, 2004, springer.

[12]: cours de compression, Dr MVogo Nogono Foseph Master 2.

[13] : Alexandre THIL- Master1 Informatique –Université de Metz.

[14] : la compression des images animées ,Sébastian Crespin- 5 décembre 2001, TER Réseaux - DESS TNI - Université Montpellier II.

[15]: la compression de son, unil, université de Lausanne.

[16]: http:// www.yann-ollivier.org/

[17]: http:// iict/Tcom/cours/PDF/compression.pdf/

[18]: http:// di.ens.fr/pointche/enseignenent/enste2/

[19]: http:// www.dataligence.com/

[20]: http://www.wikipedia .org/

[21] : http://www.commentcamarche.com/

Référence

[22] : http:// Unil.ch/guichet/

[23] http://www.lesitedemika.org/

24[01] : Marc Lelarge : Théorie de l'information et codage

25[02] : EL Moukhtar ZEMMOURI, Introduction à la compression de données (Codage source)

26[03] : Mvogo Ngono Joseph, Principes généraux de codage entropique d'une source

27[04] :codage de source pdf*Université de Caen - UFR de Sciences*:

28[06] : ida meng yi pu, fundamental data compression ,edition elsevier

29[07] :http://www. Codage d'une source.mht

30[08] : Steven Pigeon, Contributions à la compression de données (Université de Montréal)

31[09] :David Salmon,variable-lenght codes for data compression. Edition Springer,LONDON,2007.

32[10] : David Salmon,DATA compression .Edition Springer,NEW YORK,2004.

33[11] : David Salmon,A concis introduction to data compression. Edition Springer,LONDON,2008.

[34] :S.Maadi, Y.Peneveyre, C. Lambercy, (2002).compression de données avec pertes. Suisse : IICT [Institute for Information and Communication Technologies].

[35] : http://www.01net.com/editorial/274323/comment-ca-marche-la-compression-numerique/