REPUBLIQUE ALGERIENNE DEMOCRATIQUE ET POPULAIRE

MINISTERE DE L'ENSEIGNEMENT SUPERIEUR ET DE LA RECHERCHE SCIENTIFIQUE

UNIVERSITE MOULOUD MAMMERI DE TIZI OUZOU

FACULTE DE GENIE ELECTRIQUE ET INFORMATIQUE

DEPARTEMENT INFORMATIQUE

Mémoire

En vue de l'obtention du diplôme de master II en informatique

Option : Système informatique

Thème :

Compression de données à l'aide des automates d'états finis déterministes

Dirigé par :
Mr : S. SADOU

Réalisé par :
Mr : MOKRANE Lyes
Melle : RAMDANI Lydia

Promotion : 2012 / 2013

Dédicaces

Je dédie ce modeste travail :

- *A mes très chers parents auxquels je souhaite une longue vie pleine de bonheur*
- *A mes frères et sœurs ainsi qu'à toute ma famille*
- *A tous mes amis (es) sans exception aucune*
- *A toi Lyes et à toute ta famille*
- *A tous les étudiants du département Informatique.*
- *A tous mes proches et toute personne que j'aime.*

R.lydia

Dédicaces

Je dédie ce modeste travail :

- *A mes très chers parents auxquels je souhaite une longue vie pleine de bonheur*
- *A mes frères et sœurs ainsi qu'à toute ma famille*
- *A tous mes amis (es) sans exception aucune*
- *A toi Lydia et à toute ta famille*
- *A tous les étudiants du département Informatique.*
- *A tous mes proches et toute personne que j'aime.*

M.Iyes

Table des matières

Chapitre II : Méthodes de compression de données

Chapitre III : Analyse et conception

Chapitre IV : Implémentation et évaluation

Liste des figures

Liste des Tableaux

Introduction générale

L'informatique ne se contente pas seulement de résoudre des problèmes mais plutôt de les résoudre avec un maximum d'efficacité. Cette discipline connaît une véritable révolution malgré les contraintes technologique non négligeable rencontrées tous au long de son évolution. Le stockage et la transmission des données d'une façon économiques ont toujours constitué un problème de taille à cause de la limite de la capacité des supports de stockage et du débit des lignes de transmission.

Les développements technologiques et les exigences des utilisateurs entraînent le traitement de quantités de données toujours plus importantes; ainsi, depuis les débuts des technologies de l'information, le problème de l'exploitation optimale des voies de communication et des capacités de stockage est toujours resté un sujet d'actualité. C'est donc dans cette optique que la compression des données est devenue un enjeu crucial.

De multiples études ont été menées sur les algorithmes de compression, dans le but de mettre les données sous un format tel qu'elles occupent moins de volume. Une fois compressées, les données ne sont plus directement accessibles, et il est nécessaire de les décompresser pour qu'elles redeviennent intelligibles.

La compression des données consiste à réduire le nombre de bits nécessaires à la représentation de celles-ci. La compression de données met en œuvre deux processus, le premier compacte les données (les encode) et le second décompresse les données (les décode) pour qu'elles se retrouvent dans leur état initial.

Les méthodes de compression de données se divisent en deux classes : la compression avec perte d'information et la compression sans perte d'information. Dans la première classe, on tolère une certaine perte de précision en échange d'un taux de compression plus élevé ; c'est acceptable dans certaines applications, comme le traitement des images et du son, à condition que cette dégradation soit correctement gérée. Cela dit, on utilise souvent une compression sans perte d'information, qui assure une copie exacte de l'original lors du décompactage des données.

Dans ce projet nous avons adopté une nouvelle méthode de compression de données à l'aide d'automate d'états finis déterministes.

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

Chapitre I: rappelle des concepts de base et les définitions essentielles et généralités de la compression de données.

Chapitre II: dans ce chapitre nous allons présenter les principales techniques de compression de données.

Chapitre III: est consacré pour l'analyse et la conception de notre méthode, ainsi nous allons expliquer les différents modules et algorithmes à développer.

Chapitre IV: représente l'implémentation et l'évaluation de notre méthode.

Chapitre I :
Généralités sur la compression de données

I.1 Introduction :

La compression de données est un vaste sujet qui a fait l'objet de nombreux ouvrages et articles. Elle donne lieu aujourd'hui à de nombreuses recherches en raison des enjeux économiques. Elle est utilisée majoritairement dans les applications informatiques.
La compression de données est considérée comme solution qui peut être employée pour alléger une portion de problème de stockage pour faire en sorte que les données occupent le minimum d'espace pour une vitesse de restriction acceptable. Par ailleurs, la compression peut participer à l'opération de sauvegarde de données et offrir un niveau de cryptage contre des intrusions illicites puisqu'un texte initialement représenté par un code conventionnel comme le standard ASCII est converti en un code différent.

I.2 La théorie de l'information : [1]

C'est la théorie mathématique traitant des bases théoriques de la transmission et du traitement de l'information. La théorie de l'information est une approche statistique dont les premiers rudiments ont été proposés par HARLEY en 1928, mais dont les fondements n'ont vraiment été établis qu'en 1948 par le mathématicien CLAUDE, SHANNON dans son livre intitulé **«** théorie mathématique de la communication ».

Cette théorie s'intéresse à la mesure de la quantité d'information, à la représentation de cette information, dite aussi codage, ainsi qu'aux systèmes de communication qui la transmettent et la traitent. Ce codage peut ainsi se référer à la conversion du son et d'images en signaux électromagnétiques.

Outre les télécommunications l'électronique et l'informatique, la théorie de l'information s'applique à divers domaines comme la cybernétique, la linguistique et la psychologie.

I.3 Quelques éléments de la théorie de l'information :

I.3.1 Sources d'informations :

Nous concéderons ici des sources discrètes d'information. C'est-à-dire des sources qui produisent un nombre fini d'élément s appelaient symboles de sources et noté S_i.

Exemple :

Source d'information clavier d'ordinateur.

I.3.1.1 Définition :

L'ensemble des symboles de la source constitue un alphabet.

Exemple:

L'alphabet d'un clavier de téléphone portable est l'ensemble {0, 1, 2, 3, 4, 5, 6, 7, 8, 9, *, #}

I.3.1.2 Source discrètes à mémoire : [2]

L'hypothèse de symboles indépendants n'est pas toujours vérifiée. Prenons exemple à un texte en français: la probabilité d'avoir la lettre "u" est très grande si la lettre précédente est "q". Cette probabilité d'avoir conditionnelle est noté P (u/q). Les pixels d'une image sont aussi fortement corrélés car il est rare que la couleur ou la luminance d'un pixel change complètement par rapport aux pixels voisins.

I.3.1.3 Source discrète sans mémoire : [2], [3]

Pour caractériser les sources, de nombreux termes sont empruntés de la description du langage courant. Une source dispose d'un "alphabet" constitué d'éléments ou symboles ou caractères $\{x_1, x_2, x_3,..., x_k\}$ .K est la longueur de l'alphabet. Ces symboles sont associés pour constituer un message. Emettre un message revient à émettre une succession de symboles appartenant à une source. Chaque symbole x_i de l'alphabet a une probabilité d'utilisation p_i. Pour simplifier le problème, une catégorie de sources est plus simple à modéliser: celle des sources pour lesquelles la probabilité d'émission d'un caractère est indépendante de ce qui a été émis avant ou sera émise après. C'est ce qui définit une source sans mémoire.

I.3.2 La quantité de l'information : [4], [5]

L'information produite par une source doit être quantifiée. Pour cela on à d'abord définie la quantité d'information contenue dans un symbole. Partant de l'idée que si la source produit toujours le même symbole. Ce symbole est prévisible et ne contient pas d'information. A l'opposé, plus se symbole est rare, plus il contient de l'information. La notion de l'information donc liée à la probabilité d'apparition $\mathbf{P(S_i)}$ des symboles SI pour une source dont les symboles sont statiquement indépendants (source discrète sans mémoire) la quantité d'information d'un symbole est définie par:

$$I(S_i) = \log 2\left(\frac{1}{\mathbf{p(si)}}\right)$$

Cette définition peut être généralisée à la quantité d'information d'un message. C'est-à-dire une suite de symboles, en remplaçant **$P(S_i)$** par **P(M)** la probabilité du message, si un message est composé de 'm' symboles statistiquement indépendant, sa quantité d'information équivaudra à la sommes des quantités d'information de chaque symboles car:

$$\mathbf{P(M)= P(S_1)* P(S_2)*.....*P(S_m).}$$

Où: **P(M)** est la probabilité du message en question, soit produit.

Donc: La quantité d'information d'un message **M** de m symboles est :

$$\mathbf{Im} = \sum_{\mathbf{N}}^{\mathbf{1}} \mathbf{Ii}$$

La quantité d'information d'un message désigne également le nombre de symboles binaire nécessaires pour représenter ce message.

A partir des remarques suivantes:

- Ø La quantité d'information d'un symbole est d'autant plus grande que celui ci est peu probable.
- Ø la quantité d'information de deux symboles successifs est la somme de leur quantité d'information.

La quantité d'information notée I est une fonction qui doit ainsi avoir les propriétés suivantes:

1) **I ()** est une fonction continue de la probabilité $\mathbf{P_i}$.

2) **$I(P_k)$** crois si p_k décroit donc **$I(P_k)$** est une fonction décroissante de $\mathbf{P_k}$

3) **$I(P_k \text{ ET } Pj) = I(P_k) + I(P_j)$**

4) un symbole qui possède une quantité d'information nulle: **$I(P_K=1)=0$**

Une fonction mathématique remplit les conditions : 1, 3 et 4 : $\log(P_K)$

Pour obtenir la propriété 2 il suffit de prendre **$\log(P_K) = -\text{Log}(1/P_K)$.**

La quantité d'information d'un symbole X_K de probabilité P_K est ainsi définie par Shannon comme suit:

$$\mathbf{I\ (X_K) = -Log\ (P_K) = Log\ (1/P_K)}$$

I.3.3 L'entropie : [6]

En générale, l'entropie mesure le degré de désordre dans un system, en théorie de l'information, il indique le poids d'information mathématique que porte un message, et donc sa compressibilité théorique.

L'entropie (H) d'un message est l'information moyenne contenue par chaque symbole, elle est donnée par la relation suivante :

$$\mathbf{H(x) = \sum_{i=1}^{n} p(i) \log_2 \left(\frac{1}{p(i)}\right) = -\sum_{i=1}^{n} p(i) \log_2 p(i)}$$

I.3.3.1. Entropie maximale :

L'entropie est maximale $\mathbf{H_{max}(x)}$ si la probabilité d'apparition des symboles est égale à $\frac{1}{n}$.

I.3.3.2. Utilité de l'entropie :

D'un strict point de vue quantitatif, on ne peut que constater que la plupart des langues comprennent beaucoup moins de mots que les possibilités offertes par l'alphabet, ainsi les codages traditionnels, qui représentent ou cherchent à représenter toutes les combinaisons théoriques, se révèlent extrêmement simplistes et peu performants. Le codage ASCII sur 7 bits, par exemple, fournit plus de 4 000 milliards de possibilités pour coder les mots de 8 lettres, alors que les dictionnaires de la langue française comptent de moins de 3 000…

En tenant compte des caractéristiques d'entropie des données à traiter, il est donc possible de définir des codages plus performants. Les travaux de Shannon et ses collaborateurs ont conduit à développer des codages fondés sur la fréquence d'apparition d'une information.

I.3.4 La redondance :

La source X est dite sans redondance, si tous les événements de X apparaissent avec la même probabilité ($p(x_1) = p(x_2) = ... p(x_n)=1/n$).

La redondance (**R**) d'une source caractérise la différence qu'il existe entre la quantité d'information que transportée cette source et la quantité d'information que cette source transporterait si tous ses symboles étaient équiprobables et indépendants. On a:

$$R= 1-(H(x)/H_{max}(x))$$

R est compris entre 0 (les symboles de la source sont indépendants et équiprobables) et 1 (l'entropie de la source est nulle).

Exemple :

Dans la langue française la lettre « e » se trouve plus de fois que la lettre « f » dans les messages de texte donc « e » à plus de redondance que « f ».

I.4 La modélisation :

C'est l'opération qui consiste à extraire à partir du flux de données en entrées, les informations qui représenteront ce flux dans le codeur, et à base des quelles ce dernier effectuera le codage.

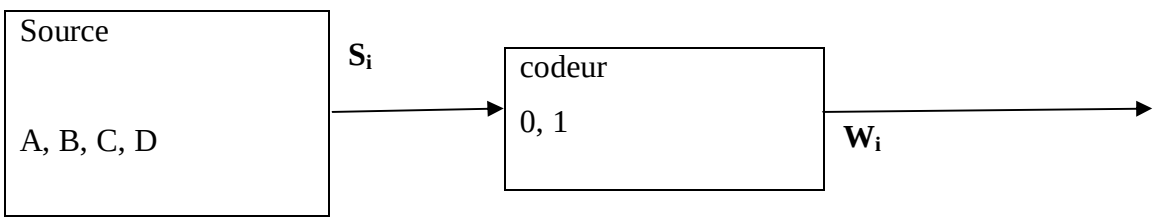
I.5 Codage et code :

I.5.1 Définitions :

C'est une fonction qui fait correspondre à tout symbole ou groupes de symboles du flot de données à compresser, par des symboles plus compactes.

I.5.2 Codage de la Source :

Le codage de source consiste à représenter les symboles de source S_i par des mots W_i appelés **Mots de code**. Le code est la table de conversion entre les symboles S_i et les mots de code W_i qui sont souvent des mots binaires. La conversion est effectuée par un **codeur**. La longueur d'un mot de code est le nombre de symboles binaires qu'il contient.



I.5.3 Propriétés d'un code :

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

- Ø Tous les mots du code peuvent être distingués.
- Ø Le décodage ne donne lieu à aucune ambigüité.
- Ø Il doit être préfixé: Aucun mot du code n'est un sous mot initial d'un autre.
  Cette propriété permet d'assurer un décodage unique.

I.5.4 Les types de codes (selon la taille) : [7], [8]

Les différents types de code peuvent se présenter sous les formes suivantes:

1)- code à longueur fixe(FLC) : (fixed length codes) est un code qui associé à touts les messages de la source un nombre constant de bits(les événements de la source sont codés sur le même nombre de bits).

2)-code à longueur variable(VLC) : (variable length codes) est un code qui associé les messages de la source à un nombre de bits).

Les codes à longueur variable sont souvent choisit pour la compression des données, parce qu'avant toute opération d'enregistrement des données assigne des codes plus court pour les symboles plus fréquents, et des codes plus long pour des symboles plus rares.

Par exemple :

Considérons le code à longueur variable suivant (0, 100, 101, 110, 111) avec des longueurs de code (1, 3, 3, 3, 3) pour l'alphabet (A, B, C, D, E), et la chaine source BAAAAAAAC avec des fréquences pour chaque symbole (7, 1, 1, 0, 0).la moyenne en nombre de bit est :

$$\boldsymbol{l = \frac{1 * 7 + 3 * 1 + 3 * 1}{9} = 1.4 bits/symbole}$$

C'est presque un enregistrement avec la moitie de bits comparé avec 3 bits/symbole si on utilise un code de longueur fixe à 3 bits.

Le plus court code est attribué au plus fréquent symbole dans la chaine source.

Comme ça le code est le meilleur de point de vue compression.

Exemple d'application des VLCs :

Le code morse assigne les petits codes pour les lettres les plus fréquentes et les codes plus long pour les moins fréquentes.

Le domaine de fabrication de microprocesseurs. Le principe est de chercher une instruction en mémoire et de l'exécuter, ces instructions sont généralement codés avec des tailles fixes, cela facilite leur manipulation par le microprocesseur. Mais les processeurs modernes optent pour une taille d'instructions variables. Il est alors possible de réduire la taille des programmes en utilisant des petits codes pour les instructions les plus fréquentes dans des programmes (exemple: l'instruction de changement : load).cela réduit aussi la taille physique du microprocesseur et consommation d'énergie.

Le standard **ISBN** (International Standard Book Number), est un nombre unique qui est attribué aux livres. Une partie de ce code désigne le code de pays de publication, la longueur de ce code varie entre 1 et 5 digits, et il assigne des codes pour les pays qui publient plus.

Le code d'appel téléphonique international. La recommandation standard internationale attribue un code de longueur variable pour les pays selon la quantité d'appareil téléphonique que ce pays possède.

I.5.5 Déchiffrabilité d'un code : [7], [8]

Un code C sur un vocabulaire V est dit uniquement déchiffrable (on dit parfois non ambigu) si seulement si, pour tout $x=x_1...x_n \in V^+$. Il existe au plus une séquence $c=c_1\ldots c_m \in C^+$ tell que $c_1\ldots c_m = x_1\ldots.x_n$.

Propriété :

Un code C sur un vocabulaire V est uniquement déchiffrable si et seulement si pour toutes séquence $c=c_1\ldots c_n$ et $d=d_1\ldots.d_m$ de C^+.

$C=d \longrightarrow (n = m \quad et \quad 1 <= i <=n, c_i=d_i)$.

Théorème 1(Kraft) : il existe un code uniquement déchiffrable sur un vocabulaire V dont les mots $\{c_1\ldots c_n\}$ sont de longueur $l_1\ldots l_n$ si et seulement si :

$$\sum_{i=0}^{n} \frac{1}{|Vli|} <= 1$$

I.5.6 Propriété de préfixe : [7], [8]

On dit qu'un code sur un vocabulaire V à la propriété du préfixe (on dit parfois qu'il est instantané, ou irréductible) si et seulement si pour tout couple de mots de code distincts (c_1, c_2), c_2 n'est pas un préfixe de c_1.

Grace à la propriété de préfixe, on peut déchiffrer les mots d'un tel code dés la fin de la réception du mot (x_n), ce qui n'est pas toujours le cas pour les codes uniquement déchiffrables : ex, si V=0, 01, 11 et si on reçoit le message m=0011111111111111111…il faut attendre le symbole qui suit 0 pour pouvoir savoir déchiffrer (0 ou 01 ?).

Remarque :

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

I.5.7 Classification de codage :

On peut donner trois types de codage :

1)-Codage de canal : est utilisé pour le cryptage des données à transmettre pour des raisons de sécurité d'information.

2)-Codage correcteur d'erreurs : on les utilise pour la correction des erreurs de transmission.

3)-Codage de la source : est utilisé pour la compression de données.

I.6 La compression et la décompression de données :

I.6.1 La compression :

I.6.1.1 Définition :

La compression de données (en anglais data compression ou data compaction) traite de la manière dont on peut réduire l'espace nécessaire à la représentation d'une certaine quantité d'information.

Elle a donc sa place aussi bien lors de la transmission que lors du stockage des données. Elle est très utile pour plusieurs applications informatiques.

I.6.1.2 Intérêt :

De nos jours, la puissance des processus augmente plus vite que les capacités de stockage, et énormément plus vite que la bande passante des réseaux (car cela imposerait d'énormes changements dans les infrastructures de télécommunication).

Il ya donc un déséquilibre entre le volume des données qu'il est possible de traiter, de stocker, et de transférer.

Par conséquent, il faut donc réduire la taille des données. Pour cela, il faut exploiter la puissance de processeurs pour pallier aux insuffisances des capacités de stockage en mémoire et des vitesses de transmission sur les réseaux.

I.6.2 La décompression :

En utilise le verbe décompacter (to unpack), c'est la restauration des éléments compressés ou compactes à leurs format d'origine. Donc c'est le processus inverse de la compression.

I.6.3 Le schéma général de compression/décompression: [9]

Il montre de manière globale l'opération de compression décompression de données.

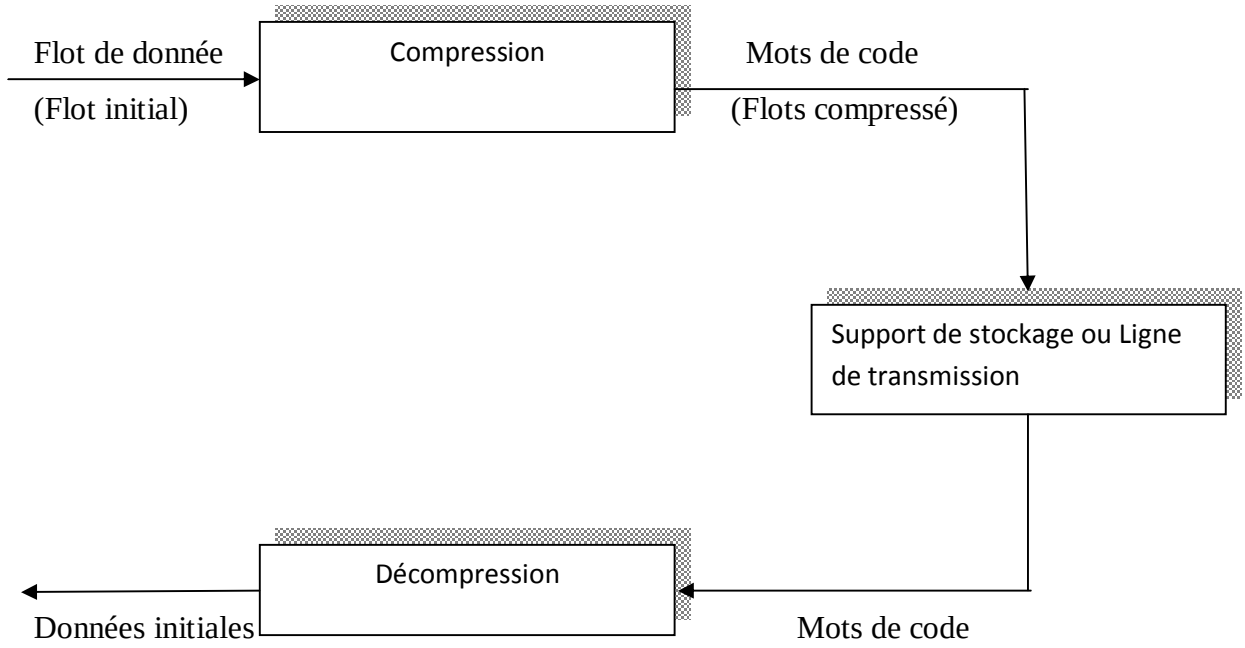


Figure I.1 : Schéma général de compression/décompression

I.6.3.1 Le module de compression (compresseur, encodeur) :

Il reconnait les messages délivrés par la source, code ces messages pour avoir une suite de codes en sortie.

I.6.3.2 Le module de décompression :

Il restitue les données initiales à partir des codes délivrés par une ligne de transmission ou par un support de stockage.

I.6.4 Les types de compression de données :

Dans le Domain de la compression, il existe plusieurs façons de comparer les types de compression. Pour cette raison, nous allons voir comment classifier les types de compression de données.

I.6.4.1 La compression physique : [10]

Elle agit directement sur les données, il s'agit de regarder les données redondantes d'un train de bits à un autre.

I.6.4.2 La compression logique :

Elle est effectuée par un raisonnement logique en substituant une information par une information équivalente (recorder les données dans une représentation différente plus compacte contenant la même information).

Exemple : Remplacer un symbole alphabétique, numérique ou binaire en un autre. 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 chaine « United State of America » et garde la même signification. La substitution logique ne fonctionne qu'au niveau du caractère ou plus haut et est basée exclusivement sur l'information contenue à l'intérieur des données.

I.6.4.3 Compression symétrique/asymétrique : [11]

Dans le cas de la compression symétrique, la même méthode est utilisée pour compresser et décompresser l'information, il faut donc la même quantité de travail pour chacune de ces opérations. C'est ce type de compression qui est généralement utilisée dans les transmissions de données. La compression asymétrique demande plus de travail pour l'une des deux opérations, la plupart des algorithmes requiert plus de temps de traitement pour la compression que pour la décompression. Des algorithmes plus rapides en compression qu'en décompression peuvent être nécessaires lorsque l'on archive des données auxquelles on accède peu souvent (pour des raisons de sécurité par exemple).

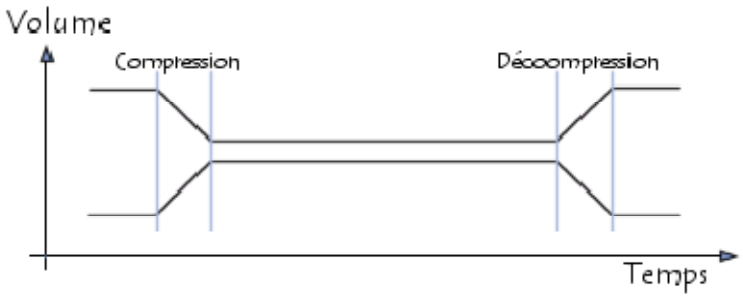


Figure I.2 : Compression de type symétrique

I.6.4.4 La compression statistique / numérique :

On peut encore distinguer les algorithmes qui travaillent au niveau statistique et ceux qui opèrent au niveau numérique.

Pour les premiers types de compression La valeur des motifs ne compte pas. Ce sont les probabilités qui comptent, et le résultat est inchangé par substitution des motifs tandis que les seconds, Les valeurs des motifs influents sur la compression (par exemple JPEG), et les substitutions sont interdites.

I.6.4.5 La compression adaptative, semi adaptative et non adaptative :

La compression non adaptative :

Certains méthodes statique utilisent le codage Huffman sont conçues pour compresser seulement des types spécifiques de données. On utilise des codages non-adaptatifs qui contiennent un dictionnaire statique de chaines de caractère prédéfinies qui sont connues comme apparaissant à de grandes fréquences dans les données à encoder. Par exemple, un codeur non-adaptatif conçu spécifiquement pour compresser la langue française contiendra un dictionnaire avec des chaines de caractères telles que "et", "mais", "le", car ces chaines apparaissent très fréquence dans les textes en français.

La compression adaptative :

A l'inverse n'intégrera pas de données relatives à la fréquence d'apparitions des données à compresser. Des compresseurs adaptatifs comme LZW ou Huffman dynamique déterminent la dépendance des données en construisant leur dictionnaire à la volée. Ils n'ont pas de listes prédéfinies de chaines de caractères par exemples mais les construisent dynamiquement à l'encodage.

La compression adaptatif est capable de s'adapter à n'importe quelles données d'entrées et de retourner une sortie avec le taux de compression le meilleur possible. C'est une des principales différences avec les compressions non-adaptatives qui sont capable d'avoir des codages efficaces uniquement avec un type de données d'entrées très restreint pour lequel ils ont étés conçus.

La compression semi adaptative :

Un mélange de ces deux méthodes. Un encodage semi-adaptatif fait un premier passage sur les données pour construire le dictionnaire et un second passage pour effectuer l'encodage. En utilisant cette méthode, un dictionnaire optimal est construit avant qu'un quelconque encodage soit effectué.

Enfin le critère de classification le plus pertinent est basé sur la perte des données.

I.6.4.6 La compression sans perte et la compression avec perte :

I.6.4.6.1 La compression sans perte (non destructive) : [12]

Une compression est dite « sans pertes » lorsqu'on peut obtenir les données initiales compressées sans qu'il ait eu aucune altération.les programmes informatiques, codes sources et rapports sont toujours encodés de cette façon.

On utilise la compression sans perte pour compacter les sources de données ou toute perte d'information est une destruction de contenu de la source.

Quelque domaine qui demande ce type de compression:[11]

- les informations provenant des sondes spéciales dont les données qui les représentent sont sensible au contexte, tout changement (suppression de donnée) engendre une modification radicale dans les valeurs scientifique.
- L'imagerie médicale, ou pour faire une analyse médicale exacte on a besoin d'une image originale qui donne les informations réelles, parce que le moindre détail peut changer le résultat de l'analyse...

Remarque :

La compression sans perte utilise des algorithmes qui compressent les données sans les dégrader. Autrement dit, après la décompression des données, celles-ci sont identiques à celles utilisées pour la compression

Ces algorithmes de compactage sont utilisés pour compresser tous types de données : des données textuelles, des images, du son, des programmes, etc.

Les trois principales catégories d'algorithme sans perte sont :

- **Les Algorithmes à base de dictionnaire :**

Découpent les données en « mot » qui sont mis dans un dictionnaire, si le mot est déjà dans le dictionnaire on le remplace par son indexe si non on l'ajoute dans le dictionnaire, on espère que tous les mots finiront dans le dictionnaire pour aboutir à une compression maximale.

- **Les Algorithmes à base de transformation :**

Appliqueront plutôt une transformation sur les donnes qui mettra en évidence les répétitions, de façon à pouvoir les mieux exploiter pour la compression. Ces algorithmes sont souvent suivis d'une autre tape de compression comme méthode à base de dictionnaire ou statistique.

- **Les Algorithmes à base modélisation statistique :**

Calculent des probabilités pour aider à la compression, typiquement, on tentera d'estimer le plus précisément possible la fonction de distribution de la prochaine donnée, de façon à pouvoir générer un code efficace pour cette prochaine donnée, en théorie ils peuvent mener à une compression optimale.

I.6.4.6.2 La compression avec pertes (destructive) : [12]

Il est question de compression « avec pertes » lorsque certaines informations sont volontairement supprimées, par exemple les détails d'une image, ou les fréquences inaudibles pour un fichier sonore. Certains éléments sont quasiment imperceptibles par l'humain (par exemple les très basses ou hautes fréquences audio), 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 ».

Exemple:

Un algorithme avec perte pourra par exemple d'éliminer le bruit d'un signal et n'encoder que le signal. Nous avons une perte parce que le bruit contient aussi de l'information. Mais puisque c'est une information qui peut détruire le signal alors sa suppression apporte un avantage.

Remarque :

Les algorithmes de compression destructrice sont utilisés pour compresser des données graphiques, audio et vidéo. Ils ne pourraient être appliqués sur des données textuelles ou sur celles d'un programme au risque de les rendre illisibles ou inexécutables.

I.7 Topologie de données à compresser :

I.7.1 Le texte :

De manière générale, toute information structurée en ASCII ou composée de caractères alphanumérique. Dans un traitement de texte et en PAO (publication assistée par ordinateur), partie principale d'un document, par opposition aux titres, tableaux, figures, notes de bas de page et autres éléments.

I.7.2 Le son :

Un son peut être défini comme une vibration généré mécaniquement, transmise généralement par 'aire sous forme d'ondes qui aboutissent au tympan de l'oreille, avant d'être interprété par le cerveau c'est donc un signal que l'on peut représenter sous forme d'une courbe mathématique indiquant l'intensité en fonction du temps qui s'appelle un signal analogique ce signal doit encore être numérisé pour pouvoir être exploiter par l'ordinateur pour ce la, il est échantillonné, c'est-à-dire découpé dans le temps, par une carte son.

I.7.3 L'image : [13]

C'est une représentation plane d'un objet tridimensionnel, perçu par l'œil humain, ou plus généralement un capteur dont le fonctionnement est semblable (exemple d'objet : une scène, un portrait, une échographie, une observation astronomique….). Elle peut être décrite sous la forme d'une fonction I(x, y) de brillance analogique continue définie dans un domaine borné, tel que x et y sont les coordonnés spatiales d'un point de l'image et i une fonction d'intensité lumineuse ou de couleur.

I.7.3.1 La définition :

Pour une image numérisée, on utilise une autre notion qui est la définition, laquelle peut s'exprimer par plusieurs valeurs, soit le nombre de point par ligne soit le nombre de lignes, soit le produit des deux.

I.7.3.2 La numérisation :

La numérisation d'une image est la conversion de celle-ci de son état analogique en une image numérique représenté par une matrice bidirectionnelle de valeur numérique f(x, y)

Ou : x, y : coordonnées cartésiennes d'un point d'image.

I.7.3.2.1 Modes de numérisation :

Mode de niveau gris :

Dans ce mode la valeur du rouge (R), du vert (V) et du bleu (B) est la même, donc codé sur n=8 bits, nous pouvons donc définir 256 niveaux de gris, allant du niveau 0 (noir) au niveau 255 (blanc).

Mode couleur RGB :

RGB (Red-Green-Blue) ce triplet permet de définir une couleur : chaque couleur peut prendre une valeur de 0 à 255, indiquant son intensité. Par exemple, un triplet (255, 0, 0) représente la couleur rouge foncé. De même, (0, 0, 0) représente le noir et (255, 255, 255) le blanc. C'est ce que on appelle la synthèse additive (rouge+vert+blanc) à la quelle notre œil est sensible, $(2^8)^3$ couleurs, soit pratiquement ce que l'œil humaine peut voire. Cette échelle additive s'imagine facilement en ajoutant des couleurs à la couleur noire. Donc plus on ajout de couleurs (R, V, B), plus on ce rapproche du blanc. Elle est utilisée pour des écrans.

Elle s'oppose à l'échelle soustractive, où l'on soustrait des couleurs à la couleur blanc (CMYK : Cyan, MAgeta, YELLOW, black ; soit CMJN en français). Cette quadrichromie est le principe utilisé par les imprimantes couleur, la photographie. On trouve également des autres modèles, comme le YUV (luminance et chrominance), utilise pour les téléviseurs (et le format JPEG), et HSV (ou HSB) : HUE, saturation, Brightness.

I.7.4 La vidéo :

Du latin vidéo « je vois » technique de transformation d'images animées en signaux électriques (signaux vidéo), destinée à permettre leur diffusion ou leur enregistrement. Les premiers pas en terme de compression ont été fait a partir du JPEG et on donné naissance au M-PEGE (Moving JPEG), qui consiste à compresser chaque image au format JPEG. Ceci a amené à coder par JPEG la première image, et à coder les différences entre chaque pair d'images successives par JPEG également.

I.8 Les format de fichiers de données :

I.8.1 Les formats graphiques : [14]

BMP (BitMap) :

Le format BMP est le format par défaut du logiciel Windows. C'est un format matriciel.

Les fichiers BMP sont constitués d'une image de type Bitmap. Ce format a été crée par Microsoft.

Ce format existe sous les environnements Microsoft Windows et OS/2, sous différentes versions.

La taille maximale d'une image est de 64k*64k (en point).

La couleur est codée sur 1 bit, 8 bit. L'image peut donc avoir 2, 16, 256 ou 16 millions de couleurs. Les données peuvent être enregistrées telles quelles ou compressées suivant l'algorithme RLE.

GIF (Graphical Interchange Format) :

Le format GIF est un format-propriétaire qui a ouvert la voie à l'image sur le Word Wide Web.

Conçu à l'origine par la compagnie H$R block, la renommée du format GIF est due au réseau COMPUSERVE. C'est un format de compression qui n'accepte que les images en couleurs indexées, codé sur 8 bits, soit en 256 couleurs.

Les images RGB(RVB) ou CYMK(CMJN) en milliers de couleurs doivent d'abord être converties en 256 couleurs avant d'être exportées en format GIF. Le format GIF est un format qui à l'origine n'existait que dans le monde des IBM/compatibles.

Toute fois, depuis quelques années, plusieurs logiciels Macintosh sont capables d'importer, de modifier et d'exporter des images en format GIF.

Le format GIF ne peut toutefois pas être utilisé par les logiciels de mise en page.

C'est un format qui présentement perd beaucoup de son marché suite à une bataille juridique concernant les droits d'utilisation sur internet.

JPEG (Joint Photographie Experts Group) :

Le format JPEG est un format de compression des images graphiques.

Ce format accepte les images RGB (RVB) et CYMK (CMJN). Il est sans doute le monde de compression le plus efficace qui soit, avec un bon compromis entre gain d'espace disque.

Temps de compression et décompression et qualité des images.

Ainsi une image brute de 2M0. N'occupera après conversion en JPEG que 400Ko, selon la qualité d'image voulue.

L'inconvénient des images JPEG, c'est qu'elles ne peuvent être importées directement dans un logiciel de mise en page.

Le format JPEG est un des formats les plus utilisés dans le monde du Word Wide Web.il peut être lu directement par les principaux outils de navigation pour le WWW.

JPEG 2000 :

JPEG de l'avenir, ou le format qui réunit les avantages du GIF et du JPEG.

Reposant sur la compression d'images par ondelettes, le JPEG est très attendu, il propose une compression avec ou sans perte, et tente d'unifier la méthode de décompression pour qu'un programme puisse ouvrir n'importe quel fichier JPEG (on en dénombre aujourd'hui plus de quarante variantes !).

Ce standard pourra faire appel à un procédée pour comprimer les zones réservées aux textes, et à un autre pour les zones photographiques.

PNG :

Le format PNG (Portable Network Graphics) est un format matriciel destiné à remplacer progressivement le format GIF sur Internet.

Ce dernier offre tous les avantages du format GIF (entrelacé, dégradation minime de l'image compressée, etc.) mais avec un meilleur taux de compression.

Il accepte les images en couleurs en 24 bits (millions de couleurs), contrairement au format GIF limité à 256 couleurs.

Bien qu'étant très peu répandus, certains logiciels de retouches de photographies permettent la conversion en format PNG. Citons à titre d'exemple adobe Photoshop (Macintosh et Windows) et GraphicConverter (Macintosh), Macromédia Fireworks.

I.8.2 Format vidéo : [15]

AVI (Audio Vidéo Interleaved) :

Format de fichier utilisé par Windows pour le stockage des images vidéo et du son, dans lequel les données audio, accélérant ainsi la vitesse de restitution. Dans ce format, on dit que l'image et le son sont entrelacés.

MPEG 1 :

Destinée au vidéos de faible résolution (325*288*25 images/s) et une qualité de son CD. Le débit est maximum.1.15 Mbits/s. Elle convient pour les applications multimédias.

MPEG 2 :

Pour la vidéo numérique de qualité TV (800*600*25 images/s).le débit est compris entre 4 et 10 Mbits/s. la qualité supérieure à la précédente.

MPEG 4 :

Prévu : pour les applications avec un très faible débit (4800 à 64000 bits/s) et destiné aux communications mobiles comme la visioconférence. La résolution est aussi, très faible (170*144*10 images/s).

MOV (QuickTime) :

Il est aussi basé sur l'utilisation de 5codec : le codec graphisme et photo/JPEG, le codec animation ainsi que les codecs Apple vidéo compacte. Lorsqu'on souhaite visualiser des fichiers d'extension AVI ou MOV sur un ordinateur, il faut que le codec correspondant soit installé.

I.8.3 Format audio :

WAV :

Le format WAV (Waveform audio File Format) est l'équivalent audio du format BMP. Egalement développé par IBM, il reprend le principe de la compression minimale. Pour simplifier, on peut dire que le WAV encode directement le son numérisé, sans aucune forme de compression, ce qui tend à produire des fichiers de taille conséquente. Pour cette raison, les fichiers WAV sont uniquement destinés aux sons très courts (tels que les jingles de Windows

par exemple). Pour les fichiers musicaux plus long, on lui préfère indéniablement le format MP3.

MP3 :

Le MP3 « MPEG » Audio layer 3 » est un format de compression de données audio par filtrage des données audio. Ce format permet de compresser à un taux de 1/12 les formats audio habituels (WAV ou CD audio). Il permet de faire tenir l'équivalent en fichiers de douze albums de musique sur un seul CD-ROM. De plus, le format MP3 n'altère que faiblement le son pour l'oreille humaine.

OGG :

Le format OGG (Ogg Vorbis) est une alternative au format MP3 qui est censée produire des fichiers de bien meilleure qualité pour une taille toutefois légèrement supérieur. Réclamé par les audiophiles qui jugent la compression MP3 trop drastique, il est développé en tant que logiciel libre, à l'inverse des formats MP3, AAC et WMA.

AAC :

Le format ACC, développé par le Moving Picture expert Group, a pour but de remplacer le MP3, prévu originellement pour accompagner les vidéos MPEG-1.

Celui-ci gère le son des fichiers MPEG-4 et doit offrir une qualité bien meilleure que le MP3 et gérer plus de canaux.

I.9 Evaluation des performances d'une compression :

On ce moment il existe beaucoup de méthodes de compression de données, adaptées a des structures de données différentes. Pour mesurer leurs efficacités et leurs performances, il faut s'intéresser sur des critères qui sont:

- Le taux de compression
- Le temps de compression/décompression
- Le gain de compression
- Le taux d'information
- Mesure de fidélité la subjectivité des experts du domaine
- Types de données (image, vidéo, texte)

- Mesure de perte d'information

Le domaine de compression des données (médicales, géologie, météorologie)

I.9.1 Le taux de compression :

I.9.1.1 Le degré de compression :

Le degré de compression des données se mesure par la comparaison de la taille des données à compresser à la taille des données initiales. On la nomme aussi: Quotient de compression donnée par la formule suivante:

$$\text{Quotient de compression} = \frac{\textbf{Taille des données initiales}}{\textbf{taille des données compressées}}$$

Remarque: Plus le Quotient n'est élevé, plus la méthode de compression employer est efficace donc le taux de compression est:

$$\textbf{Taux de compression} = \frac{\textbf{1}}{\textbf{Quotient de compression}} \textbf{*100\%}$$

I.9.2 Le gain de compression :

Le gain de compression représenté en %, c'est l'espace dont on a bénéficie après une opération de compression telle que plus le gain est important, plus la compression est efficace, il est donnée par la formule suivante:

Gain de compression = 100*taux de compression

I.9.3 Le Taux d'information :

Qui peut servir à noter la différence entre la qualité et la quantité d'information.

I.9.4 Mesure de fidélité :

C'est la distorsion introduite lors des deux processus de compression / décompression par rapport à un fichier source.

I.9.5 La vitesse de compression :

C'est le temps mis par l'opération compression / décompression sur le flux de donnée. Il y a plusieurs facteurs qui permettent de juger la rapidité d'un compresseur

I.9.6 Mesure de perte d'information :

C'est un critère d'évaluation des performances d'un compresseur/décompresseur de donnée qui ne peut restituer, les données originales aux quelles il a fait subir un processus de compression.

I.10 Conclusion :

Dans ce chapitre nous avons abordé les différentes notions et intérêt de la compression de données. Nous avons vu qu'il existait plusieurs manières de coder, de représenter, de compresser les différents médias .ensuite nous y avons décrit les divers types de la compression de données à savoir la compression sans perte et la compression avec perte. Enfin nous avons exposé quelques mesures de l'évaluation de la performance d'une compression le taux, la vitesse, et le gain de compression ainsi la mesure de perte d'information.

Dans le chapitre qui suit nous allons traiter les méthodes de la compression de données.

Chapitre II :

Méthodes de compression de données

II.1 Introduction :

La compression de données, de façon simplifiée, disons-nous, c'est l'ensemble des méthodes permettant à un grand volume de données d'occuper un volume moindre, sans perte d'information significative. Remarquez que ces méthodes ne conservent pas nécessairement toute l'information, mais l'information significative. Cette subtilité a son importance. En effet, elle peut mener à la compression de données sans perte, où chaque bit qui est soumis à la méthode de compression est restitué correctement au moment de la décompression, comme elle peut mener à la compression avec perte, où les données sont transformées de façon à ne conserver que l'information pertinente, et ou les données sont restituées de façon satisfaisante au moment de la décompression.

Toute méthode de compression de données s'effectue suivant un algorithme bien précis, plus au moins compliqué, sur des structures des données dont la manipulation plus au moins rapide. Nous nous intéressons à la compression de données sans perte qui utilise un ensemble d'algorithmes classé en deux catégories : les algorithmes à base statistique et les algorithmes à base de dictionnaire.

II.2 Méthodes de compression à base statistique:

Les programmes et les données numériques conventionnels des ordinateurs sont adaptés à la compression basée sur l'exploitation statique de la fréquence des symboles et des chaines de symboles.

Les algorithmes de ce type de méthodes lisent et codent un seul symbole à la fois en utilisant la probabilité de son apparition, les plus répandus sont :

- Le codage RLE.
- Le codage de Shannon-Fano.
- Le codage de huffman
- Le codage entropique

II.2.1 Codage RLE (Run-Length-Encoding): [12]

II.2.1.1 Principe:

Le codage RLE (Run-Length-Encoding), ou «codage par plage » consiste à réduire la taille d'un texte en remplaçant des suites de caractères identiques en couple de valeurs

(caractère remplacé, nombre d'occurrences).Par exemple « AAAAABBBBBB »pourrait être représenté par « 5*[A], 6*[B] ».

II.2.1.2 L'encodage sans séparateur :

Le codage le plus simple consiste à ne pas utiliser de séparateur pour représenter les plages de valeurs, la suite « AAAAA » sera remplacée par « A5 ».

Moins de caractères sont utilisés, ce qui permet un meilleur taux de compression que la version « avec séparateur », mais en contrepartie le texte en entrée ne doit contenir aucun chiffre. On peut cependant coder des données numériques en remplaçant les chiffres par des lettres (ex : 0 par a, 1 par b, …).

Cet algorithme peut dans certains cas produire en sortie un texte plus grand que le texte initial, par exemple « A » est remplacé par « 1A », « AA » par « 2A », « AAA » par « 3A »,le gain est donc négatif pour une seule occurrence, nul pour deux, et positif dans tous les autres cas.il a donc été décidé, par convention, ne coder uniquement les plages de longueur > 2, et copier tel quel les caractères dans le cas échéant.

Cette version de l'algorithme est donc idéale pour coder des images binaires,…

II.2.1.3 L'encodage avec séparateur :

Il est nécessaire d'utiliser un séparateur pour encoder des textes contenant des chiffres. Si « # » est choisi, la séquences «AAAAA » sera alors codée « #5#A ».le nombre d'octets nécessaire est alors augmenté de 2, on encodera alors uniquement les plages d'au 5 symboles.la suite « 15*A » sera codée « #15#A ».

Par la suite pour simplifier, le séparateur sera toujours écrit #, et |#| représentera le nombre d'occurrences consécutives de #.

Il existe un cas particulier ; lorsque le caractère « # » est rencontré, celui-ci doit être encodé d'une certaine façon pour que le décodeur puisse l'interpréter correctement. Le principe de l'échappement(ou doublement) de caractère normal, par exemple « #30# # ».

Remarque :

Ce type de codage est par exemple utilisé dans la majorité des algorithmes existants dans le but de réduire le nombre de caractères en supprimant les redondances.

La version sans séparateur sera donc utilisée lorsque les données à encoder ne contiennent aucune donnée numérique (il est peut être possible de remplacer les chiffres par des nombres, par exemple 0 : A, et 1 : B pour respecter cette condition). Dans tous les autres cas la version avec séparateur sera préférée (en choisissant de préférence un caractère peu ou pas présent dans le texte).

II.2.2 Le codage Shannon-Fano : [5] [16]

Ce codage est basé sur l'idée de répartir les symboles en deux groupes de valeur à peu prés équivalente, cette valeur étant la somme, dans chaque groupe, des probabilités d'apparition des symboles qu'il contient. Le groupe de gauche est appelé 0, celui de droite 1 (ce choix est arbitraire). Les groupes sont à nouveau subdivisés et nommés 0 ou 1 jusqu'à ce que la subdivision contienne plus qu'un symbole. L'arbre binaire aussi obtenu est formé de segment ou branches et de feuilles contient un caractère simple. Pour déterminer le code numérique d'un caractère donné, il faut partir du sommet de l'arbre et suivre les branches jusqu'à atteindre la feuille qui le présente. Les caractères les plus fréquents se trouvent le plus prés du sommet et requièrent donc moins de bits dans leurs transcriptions compressée.

II.2.2.1 L'algorithme de Shannon-Fano :

Un arbre Shannon-Fano est construit en fonction d'un algorithme spécifique conçu pour définir une table de codes efficace. L'algorithme est simple :

1. Pour une certaine liste de symboles, développer une liste correspondante de probabilité ou de compteurs de fréquence de façon à connaitre la fréquence relative d'occurrence de chaque symbole.
2. Trier la liste de symboles en fonction de la fréquence, en plaçant les symboles les plus fréquents en premiers et les plus rares en dernier.
3. Diviser la liste en deux parties, le total des compteurs de fréquence de la moitié gauche devant être aussi proche que possible du total de la moitié droite.
4. Affecter le chiffre binaire 0 à la moitié gauche de la liste, et le chiffre 1 à la moitié droite. Les codes de la première moitié commenceront donc tous par 0, et ceux de la deuxième moitié par 1.
5. Appliquer de façon récursive les étapes 3 et 4 à chacune des deux moitiés, subdivisant les groupes et ajoutant des bits aux codes jusqu'à ce que chaque symbole soit devenu une feuille représentant un code.

II.2.2.2 Exemple de codage selon Shannon-Fano :

Nous allons effectuer la compression de notre message « BANNANES ET ANANAS ». On utilisant l'algorithme de compression de Shannon-Fano. La première subdivision est indiquée sur la figure II.1.On remarque que dans l'arbre de codage résultant, on inscrit « 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, est donc arbitraire.

La subdivision définit deux symboles (qui sont fait des groupes de symboles), « AN » et ES<espace> BT cette subdivision est ainsi faite pour que les fréquences des deux subdivisions soient aussi équilibrées que possible.

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

9

8

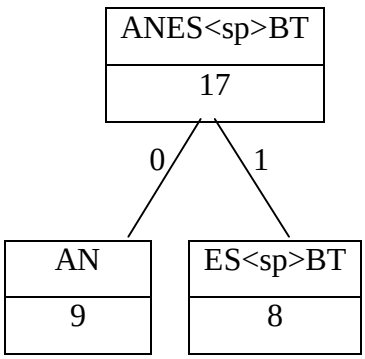
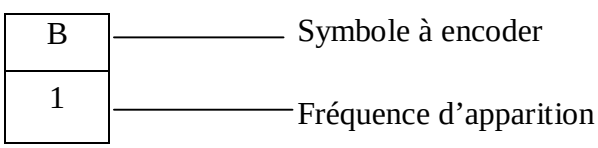


Figure II.1 : Première subdivision de Shannon-Fano

F(s) : fréquence d'apparition

S : symbole à encoder

Exemple :



La deuxième étape elle va isoler les symboles A et N, ES et « sp »BT et définir une nouvelle subdivision des symboles restant.

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

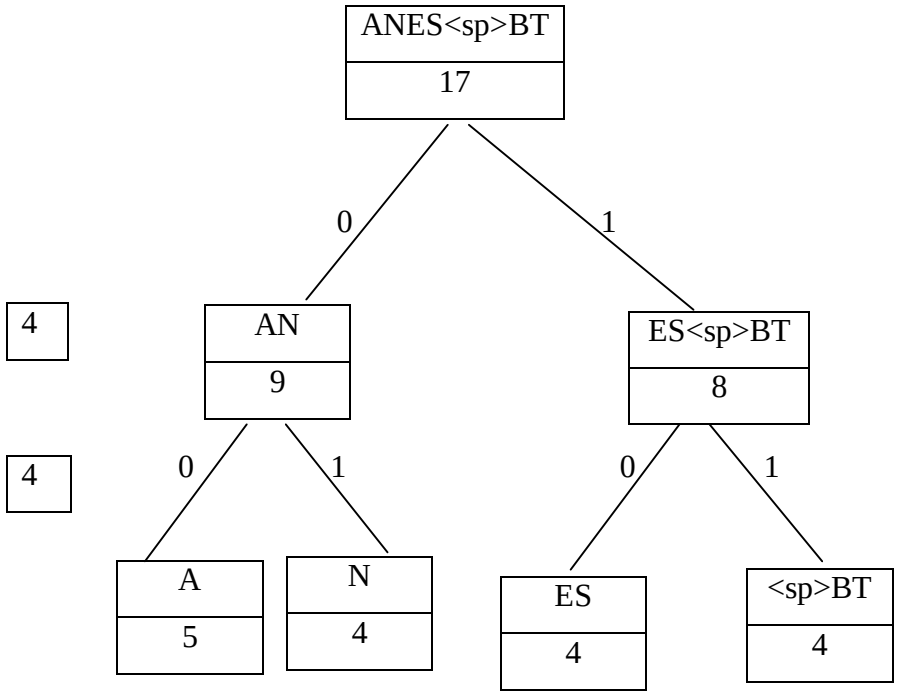


Figure II.2 : Deuxième subdivision de Shannon-Fano

La troisième subdivision isole le : E, le S et l'espace et ne laisse plus que le couple BT non résolu et le résultat finale est indiqué à la figure II.4

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

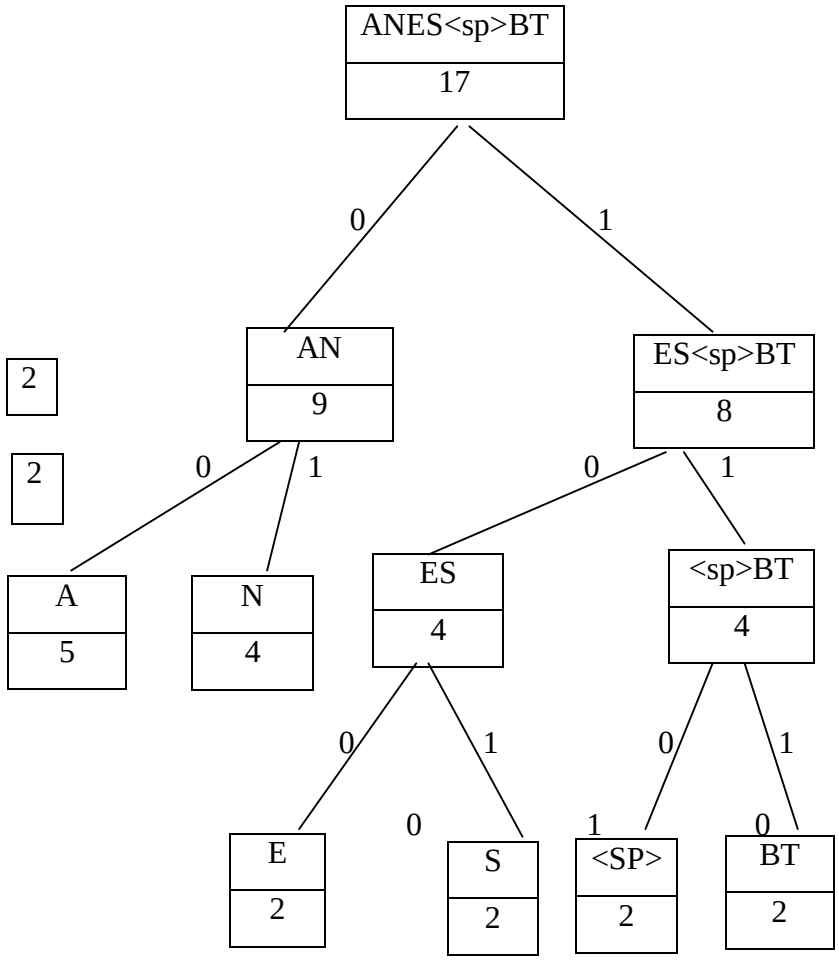


Figure II.3 : Troisième subdivision de Shannon-Fano

La dernière subdivision isole le B et le T ce qui montre la figure II.4

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

1

1

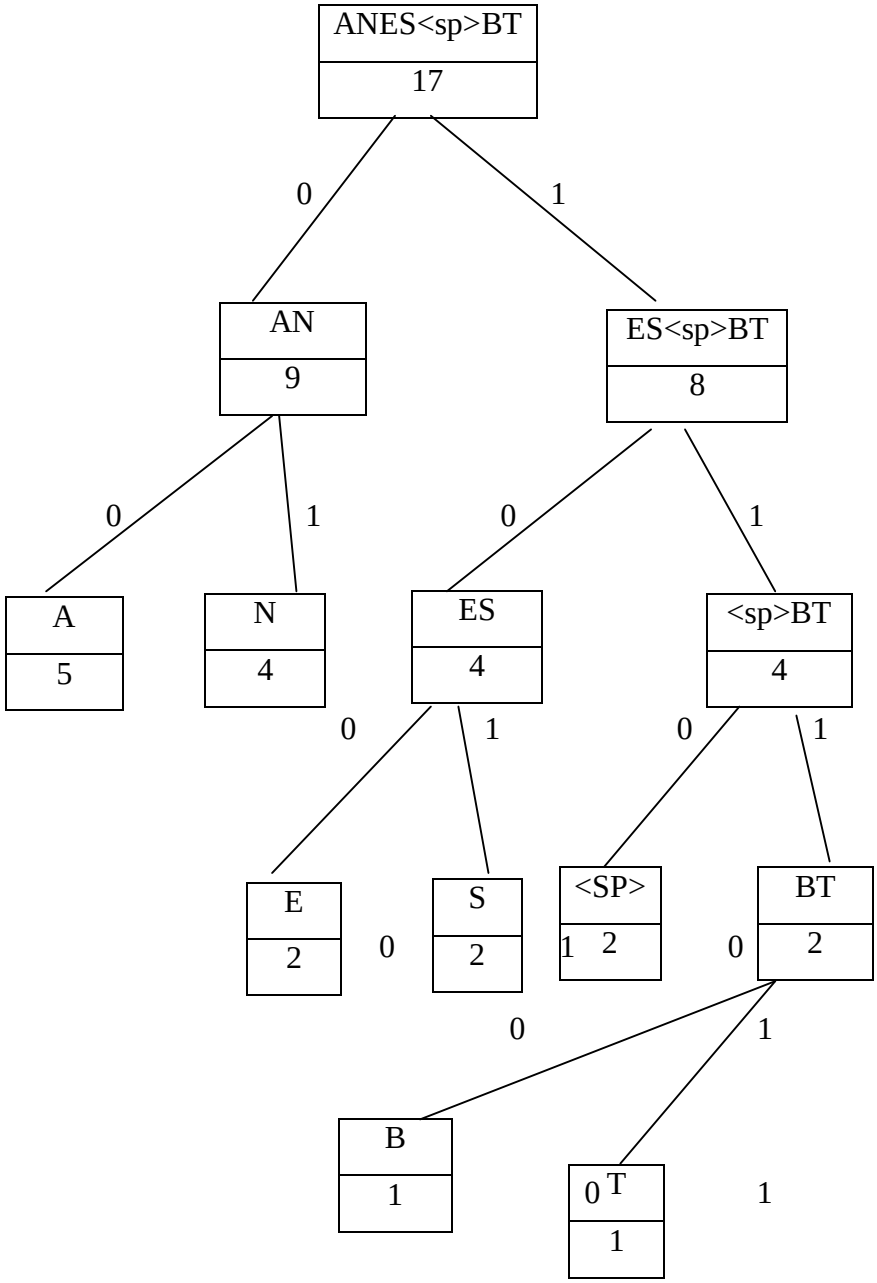


Figure II.4 : La subdivision finale de Shannon-Fano

Pour connaitre le code associé à chaque lettre, on parcourt l'arbre final de haut en bas, et l'on obtient :

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 44 bit. La redondance résultante est pratiquement nulle.

La méthode serait lumineuse s'il n'existait plusieurs manières de subdiviser une liste de symboles et s'il n'était délicat de découvrir la méthode optimale. En pratique le codage Shannon-Fano s'approche de l'optimisation idéale mais le risque existe de produire un code plus long que nécessaire.

II.2.2.3 Exemple 2 de Shannon-Fano :

symbole	**F(s)**
A	**15**
B	**7**
C	**6**
D	**6**
E	**5**

Un arbre Shannon-Fano est construit en fonction de l'algorithme qu'on a défini auparavant conçu définir une table de codes efficace.

symbole	code
A	**00**
B	**01**
C	**10**
D	**110**
E	**111**

Nous sommes sur la bonne voie car les symboles de probabilité d'occurrence la plus élevée ont le plus petit nombre de bits dans leurs codes. La formule donnant la quantité d'information d'un certaine symbole est l'oppose de logarithme base deux de la probabilité de symbole. Pour notre message théorique, la quantité d'information de chaque symbole et le nombre total de bits dans le message pour ce symbole sont donnés par la table suivante :

Symbole	F(s)	quantité	bits
A	15	1,38	20,68
B	7	2.48	17,35
C	6	2.70	16,20
D	6	2.70	16,20
E	5	2.96	14,82

L'information dans ce message totalise 85.25 bits. Si nous codons les caractères avec le jeu ASCII sur 8 bits, nous utilisant 39*8 bits, c'est-à-dire que 312 bits. Il est évident qu'une amélioration est encore possible. En utilisant le codage Shannon-Fano, le codage des mêmes données aboutit à des résultats très corrects :

Symbole	F(s)	quantité	bits	Taille SF	Bits SF
A	15	1,38	20,68	2	30
B	7	2.48	17,35	2	14
C	6	2.70	16,20	2	12
D	6	2.70	16,20	3	18
E	5	2.96	14,82	3	15

Avec le système de codage de Shannon-Fano, les 85,25 bits d'information sont codés seulement sur 89 bits. Il est claire que nous avons beaux avancés dans notre quête de méthode de codage efficace. Mais, bien que le codage Shannon-Fano soit un grand pas en avant, il a eu la male chance d'être rapidement remplacé par un système de codage plus efficace qui est le codage de Huffman.

II.2.3 Codage de huffman (arbre de huffman) : [11]

Le codage Huffman est un algorithme de compression de données sans pertes de type statique basé sur les fréquences d'apparition des caractères apparaissant dans le document initial.il a été développé par un étudiant de la MIT (Massachusetts Institute of Technology), David Albert Huffman en 1952.Cette technique est largement utilisée car elle est très efficace et on observe selon le type de données des taux de compression allant de 20% à 90% mais plus généralement entre 30% et 60% .

Le principe de compression est utilisé dans le codage d'image TIFF (Tagged Image Format File) spécifié par Microsoft corporation et Aldus corporation. La méthode JPEG (Join Photographic Experts Group) utilise aussi la compression de type Huffman pour coder les informations d'une image. (Elle utilise d'ailleurs des tables prédéfinies).

Ce procédé fait partie des méthodes de compression de type dites statiques. Cela repose sur le principe sur l'attribution de codes plus court pour des valeurs fréquentes et de codes plus longs pour les valeurs moins fréquentes. Cela est plus efficace que la représentation actuelle qui consiste, quant à elle, à utiliser une longueur fixe pour chaque symbole (exemple : un octet par caractère, code ASCCII).

II.2.3.1 Principe de l'algorithme :

L'algorithme de Huffman utilise une table contenant les fréquences d'apparition de chaque caractère pour établir une manière optimale de les représenter par une chaîne binaire (cela reprend le principe du morse qui tend à minimiser le nombre de symboles à utiliser pour les lettres les plus fréquemment employées).

On peut décomposer la procédure en plusieurs parties :

- Ø Tout d'abord, la création de la table de fréquence d'apparition des caractères dans le texte initial.
- Ø Ensuite la création d'un arbre binaire (usuellement dénommé arbre de Huffman) suivant la table précédemment calculée. (Remarque : on devrait parler plutôt de l'arborescence de Huffman.)
- Ø Enfin coder les symboles en représentation binaire optimale.

II.2.3.1.1 Table de fréquences :

Afin de construire cette table, il suffit simplement de dénombrer le nombre d'occurrences de chaque symbole S puis de calculer la fréquence f_s de chacun d'entre eux grâce à la formule suivante :

$$f_s = \frac{\text{nombre d'occurences de s}}{\text{nombre de symboles}}$$

Ensuite on trie le tableau en fonction de la fréquence d'apparition (de façon croissante) puis suivant le symbole suivant.

Supposons par exemple que nous ayons le texte suivant composé des symboles pris dans le code ASCII :

"L'algorithme de Huffman est une méthode qui permet de compresser les données".

Pour chaque symbole on calcule tout d'abord le nombre d'occurrences de chacun puis sa fréquence d'apparition suivant la formule citée précédemment.

La table de fréquences T triée est :

Caractère	Nombre d'occurrences	Fréquence d'apparition	
C	1	1/76	1.32%
G	1	1/76	1.32%
Q	1	1/76	1.32%
'	1	1/76	1.32%
A	2	2/76	2.63%
F	2	2/76	2.63%
I	2	2/76	2.63%
P	2	2/76	2.63%
U	3	3/76	3.95%
H	3	3/76	3.95%
L	3	3/76	3.95%
D	4	4/76	5.26%
N	4	4/76	5.26%
O	4	4/76	5.26%
R	4	4/76	5.26%
T	4	4/76	5.26%
M	5	5/76	6.58%
S	5	5/76	6.58%
	11	11/76	14.47%
E	14	14/76	18.42%

Tableau II.1 : Table des fréquences

II.2.3.1.2 Construction de l'arbre :

L'arbre binaire de Huffman est la structure de données qui va nous permettre d'attribuer à chaque symbole une représentation binaire optimale. Afin de construire l'arbre, on utilise la table de fréquences précédemment construite qu'on appelle T et on applique l'algorithme suivant :

Algorithme : Construction de l'arbre

Données :

– T : la table de fréquence

– Q : Une file d'attente de nœuds de l'arbre binaire. Chaque feuille est étiquetée avec un symbole et son nombre d'occurrences.

Chaque nœud interne est étiqueté avec la somme des occurrences des feuilles de sa sous arborescence.

– O : Une fonction qui à chaque nœud de l'arbre associe une valeur. Si le nœud est une feuille alors O renvoie le nombre d'occurrences du symbole, autrement O renvoie la somme des occurrences des feuilles de la sous-arborescence de nœud.

Résultat :

– A : L'arbre binaire résultant

Begin

Initialisation de Q tel que :

Q contienne les feuilles représentant les symboles de la table T

Tant que (Q non vide) **faire**

Créer un nouveau nœud z dans A tel que :

Gauche(z) = x = extraire-min (Q)

Droite(z) = y = extraire-min (Q)

O(z) = O(x) + O(y)

Insérer (z, Q

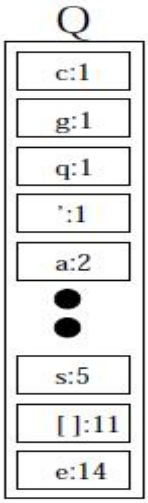
Fin

End

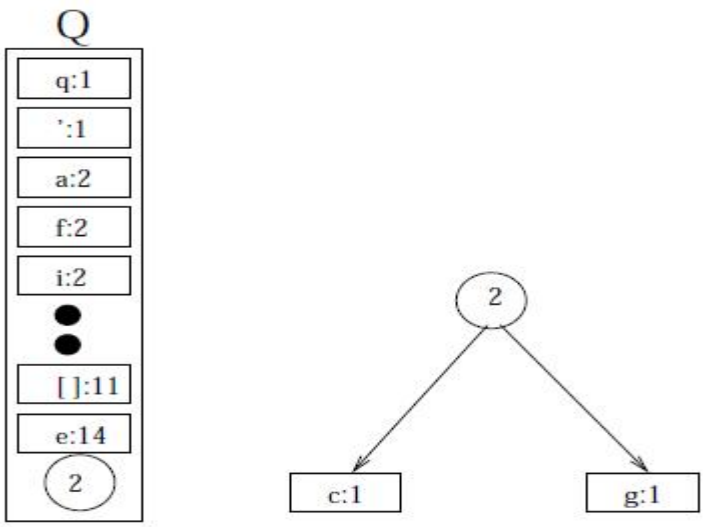
De façon informelle, on utilise une file d'attente Q dans laquelle on place les nœuds correspondants au couple [symbole : nombre d'occurrences du symbole] de tous les symboles. Ensuite on extrait de la file d'attente les 2 nœuds ayant la valeur minimale puis on crée un nouveau nœud dans l'arbre de Huffman ayant pour fils les deux sélectionnés, on rajoute ensuite le nœud nouvellement crée dans la file d'attente, et on réitère jusqu'à ce que la file soit vide.

Reprenons l'exemple précédent et appliquons l'algorithme de construction de l'arbre étape par étape.

Initialisation



Etape 1



Etape 2

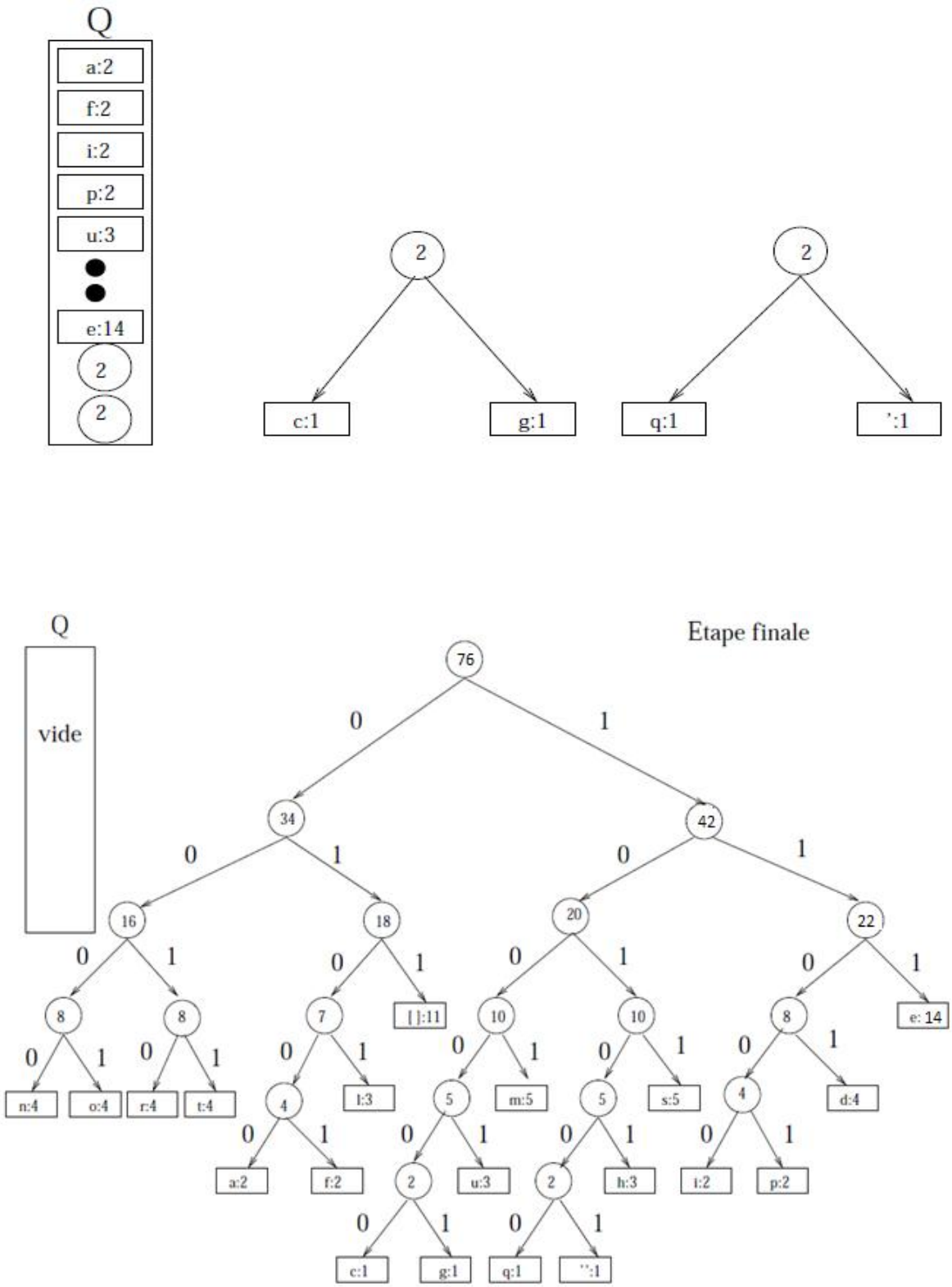


Figure II.5 : Arbre de huffman

L'arborescence de Huffman finalement construite, on étiquette les arcs de la façon suivante :

- Les arcs reliant un nœud à son fils gauche sont étiquetés par '0'.
- Les arcs reliant un nœud à son fils droit sont étiquetés par '1'.

De cette manière, chaque feuille représentant un symbole peut être redéfinie par un nombre binaire correspondant au chemin entre la racine et la feuille de l'arborescence.

Ainsi, les symboles les plus utilisés ont une représentation binaire moins importante (en termes de taille) que les symboles les moins utilisés. Ceci permet de représenter chaque symbole de façon optimale et permet de réaliser une compression des données efficacement.

Si on reprend notre exemple, on obtient la table de correspondance suivante :

Caractère	Représentation binaire (code)	Taille(en bit)	Gain(en bit)
C	100000	6	2
G	100001	6	2
Q	101000	6	2
'	101001	6	2
A	01000	5	3
F	01001	5	3
I	11000	5	3
P	11001	5	3
U	10001	5	3
H	10101	5	3
L	0101	4	4
D	1101	4	4
N	0000	4	4
O	0001	4	4
R	0010	4	4
T	0011	4	4
M	1001	4	4
S	1011	4	4
	011	3	5
E	111	3	5

Tableau II.2 : Table de correspondances

De cette manière, notre phrase exemple devient :

"L'algorithme de Huffman est une méthode qui permet de compresser les données"

l = 0101

' = 101001

a = 01000

g = 100001

...

Message compressé = 0101101001010000101100001 etc.

(Á noter que le message compressé est une suite binaire de bits alors que le message original est une suite de symboles qui peut, comme nous l'avons vu, être ramenées à une suite de bits).

- L'algorithme de Huffman est un procédé largement répandu et qui se révèle être un algorithme performant en moyenne. Il est le plus représentatif des algorithmes de compression dit de type statistique : il en existe néanmoins d'autres tels que le RLE (Run Length Encoding), ou bien VLC (Variable Length Code) dénommé souvent codage entropique. Hormis tout ces avantages, Huffman présente l'inconvénient d'être relativement ancien et d'autres algorithmes plus récent reposant sur d'autres principes tels que les algorithmes de compression à dictionnaire (exemple : LZW) arrivent a des taux de compression supérieurs en moyenne à ceux que proposent l'encodage de Huffman.

II.2.3.2 Performances :

L'intérêt de la compression est de pouvoir réduire au maximum la taille des informations originales, comme nous l'avions indiqué précédemment.

Cette mesure peut-être effectuée par le taux de compression qui est défini par la formule suivante :

$$\mathbf{p = 1 - \frac{Taille\ compréssée}{Taille\ originale}}$$

Par ailleurs, on peut aussi utiliser l'entropie qui permet de connaître le nombre minimum de bits nécessaires au codage d'un fichier.la formule de l'entropie (définit précédemment) est la suivante :

$$E = -\sum_{N}^{1}(\mathbf{Pk} * \mathbf{log2}(pk))$$

Où Pk est la fréquence d'apparition du k-ième symbole parmi les n possibles.

Analysons les performances de l'algorithme de Huffman grâce au taux de compression puis avec l'entropie sur notre exemple.

II.2.3.2.1 Performances sur l'exemple :

Taux de compression :

"L'algorithme de Huffman est une méthode qui permet de compresser les données"

Calculons la taille originale du message en bits : Le message contient 76 symboles et chaque symbole peut être représenté par le code ASCII équivalent sur 8 bits.

Taille originale = 76 *8 = 608 bits

Calculons à présent la taille du message compressé (il suffit de sommer la taille de la représentation binaire de chaque symbole), on obtient :

Taille compressée = 300 bits

On peut donc à présent calculer le taux de compression et on obtient :

$$p = 1 - \frac{Taille\ compressée}{Taille\ originale} = \frac{300}{608} = 0,5066$$

Le taux de compression dans notre exemple est proche de 50% ce qui est relativement très bon.

Entropie:

$$\mathbf{E} = -\sum_{\mathbf{N}}^{\mathbf{1}}(\mathbf{Pk} * \mathbf{log2}(\mathbf{pk}))$$

$$\mathbf{E} = -[(\tfrac{1}{76} * \mathbf{log2}\tfrac{1}{76}) + (\tfrac{1}{76} * \mathbf{log2}\tfrac{1}{76}) + \ldots\ldots\ldots\ldots\ldots\ldots + (\tfrac{11}{76} * \mathbf{log2}\tfrac{11}{76}) + (\tfrac{14}{76} * \mathbf{log2}\tfrac{14}{76})] = 3.92$$

E = 3.92

Ce qui signifie qu'il faut au minimum 3.92*76= 297.92 bits pour coder la chaîne. Avec 300 bits grâce à la méthode de Huffman on est très proche de l'optimalité.

II.2.4 Le codage entropique : [17]

Considérons une image dont chaque pixel est codé sur n bits. Cette image possède 2^n niveaux de gris différents, notés Ni (i=1,…, 2^n). Si les niveaux de gris sont indépendants et équiprobables (la probabilité pi=2^{-n}), l'entropie H de l'image est égal à $\log_2 2^n - n$ bits.
Chaque pixel de l'image porte la même quantité d'information. Dans ce cas, l'image est incompressible. Si les niveaux de gris ne sont pas équiprobable, l'entropie H de l'image est inférieure à n. le but de codage entropique est de coder les pixels à l'aide des mots de code de longueurs variables, égales à $-\log_2 pi$ bits, de manière à ce que le nombre moyen de bits par pixel soit égal à l'entropie donné par :

$$\mathbf{H=-\sum_{i=1}^{2^n} pi\ \log_2(pi) bits\ \ (H \leq n)}$$

Généralement ce codage n'est pas optimal, car la quantité $(-\log_2 pi)$ n'est généralement pas entière.

II.3 Méthode de compression de type dictionnaire :

II.3.1 Lemple-ziv-wetch: [11]

II.3.1.1 Historique et principe des algorithmes LZ** :

1) Historique :

C'est en 1977 que Jacob Ziv et Abraham Lempel fournissent une technique de compression différente de l'algorithme de Huffman, et capable de donner de meilleurs taux de compression. Ils mettent ainsi en place l'algorithme LZ77. Puis vient LZSS, version amélioré de LZ77 par Storer et Szymanski puisque la recherche des séquences dans le dictionnaire est réduite logarithmiquement.
Enfin vient l'algorithme LZ78, plus connu sous le nom LZW, amélioration faite par Terry Welch en 1984 de LZSS de par le fait que les séquences sont rangées dans une arborescence. Il porte le nom de ses 3 inventeurs : Lempel, Ziv etWelch.

2) Principe :

Le principe est fondé sur le fait qu'une séquence de caractères peut apparaître plusieurs fois dans un fichier.

L'algorithme LZW de compression consiste à émettre à la place des séquences, les adresses de ces séquences d'un dictionnaire généré à la volée.

C'est un algorithme de compression nettement plus performant en moyenne que les algorithmes statistiques puisqu'il permet d'obtenir des gains plus élevés sur la majorité des fichiers.

L'algorithme LZW se distingue des méthodes statistiques pour plusieurs raisons :

– Le fichier comprimé ne stocke pas le dictionnaire, ce dernier est automatiquement généré lors de la décompression. Il n'existe donc pas de table d'entête.

– Contrairement aux méthodes statistiques qui utilisent la probabilité de présence sur un ensemble de taille fixe de symboles, l'algorithme LZW représente un algorithme d'apprentissage, puisque les séquences répétitives de symboles sont dans un premier temps détectées puis compressées seulement lors de leurs prochaines occurrences. Le taux de compression est dépendant de la taille du fichier. Plus la taille est importante, plus le taux de compression l'est aussi.

– Il permet le compactage à la volée, puisqu'il n'y a pas à lire le fichier au préalable, il compresse les séquences de symboles au fur et à mesure.

II.3.1.2 Algorithme de compression :

1) Le principe de l'algorithme LZW de compression :

L'objectif de l'algorithme de compression LZW est de construire un dictionnaire où chaque séquence sera désignée par une adresse dans celui ci. De plus chaque séquence ne s'y trouvant pas y est rajoutée. Au final, on se retrouve avec une suite d'entiers, des adresses pointant vers une séquence contenue dans le dictionnaire.

Le dictionnaire est donc un tableau dans lequel sont rangées des séquences de symboles de taille variable, repérées par leurs adresses (leur position dans le tableau). La taille de ce dictionnaire n'est pas fixe et les premières adresses de 0 à 255 du dictionnaire contiennent les codes ASCII. Les séquences ont donc des adresses supérieures à 255.

Voici l'algorithme de compression :

Algorithme : L'algorithme LZW de compression

Données :

– Le dictionnaire des symboles rencontrés : Dico

– Le fichier à compresser : Fichier

Résultat :

– Le fichier compressé : Fichier

```
Début
    s=premier octet du fichier
    Tant que le fichier n'est pas à sa fin
        t=octet suivant
        u=concaténation(s, t)
        si (u appartient Dico) s=u
        sinon
            Ajouter (u) dans Dico
            Écrire adresse de s
            s=t
        fin si
    Fin tant que
        Écrire adresse de s
Fin
```

Exemple d'utilisation de l'algorithme LZW de compression :

A l'aide de l'algorithme LZW, nous voulons compresser la séquence suivante :

« SISI-ET-ISIS »

On lit donc le premier caractère soit 'S'. Puis on lit le suivant : 'I'. On forme la séquence u=s+t soit 'SI', puisque celle ci n'est pas présente dans le dictionnaire, on l'ajoute dans le dictionnaire. Cette séquence aura l'adresse 256, première séquence située dans le dictionnaire après la table ASCII. Enfin on attribue l'adresse de S (valeur ASCII de 'S') dans le fichier soit 83 ou (1010011 en binaire). On continue ensuite avec le caractère suivant.

La compression effective a lieu lorsqu'on rencontrera pour la seconde fois une paire ('SI' par exemple) déjà présente dans le dictionnaire. Dans ce cas on émettra l'adresse de la séquence

'SI' et non l'adresse de la séquence 'S' et de la séquence 'I', on ajoutera par la suite dans le dictionnaire la séquence de 3 caractères (soit 'SIS' dans notre exemple).

Voici un tableau résumant les opérations effectuées sur l'exemple lors du déroulement de l'algorithme LZW de compression :

	Phase 1	Phase 2	Phase 3	Phase 4	Phase 5	Phase 6
Valeur de s	S	I	S	SI	-	E
Valeur de t	I	S	I	-	E	T
Valeur de u	SI	IS	SI	SI-	-E	ET
U appartient au Dico ?	Non	Non	Oui	Non	Non	Non
Ajout dans le Dico	SI	IS		SI-	-E	ET
Ecrire adresse de	S soit 83	I soit 73		SI soit 256	-soit 45	E soit 69

	Phase 7	Phase 8	Phase 9	Phase 10	Phase 11	Phase 12
Valeur de s	T	-	I	IS	I	IS
Valeur de t	-	I	S	I	S	
Valeur de u	T-	-I	IS	ISI	IS	
U appartient au Dico ?	Non	Non	Oui	Non	Oui	
Ajout dans le Dico	T-	-I		ISI		
Ecrire adresse de	T soit 84	- soit 45		IS soit 257		IS soit 257

Tableau II.3 : Les opérations effectuées lors de déroulement de l'algorithme LZW de compression

Voici le dictionnaire résultant de l'algorithme :

adresse	séquence
0..255	ASCII
256	SI
257	IS
258	SI-
259	-E
260	ET
261	T-
262	-I
263	ISI

Enfin voici la suite d'adresse contenu dans le fichier compressé :

83.73.256.45.69.84.45.257.257

On peut ajouter que les adresses ne sont pas de taille maximale dès le départ de l'algorithme. Elles sont incrémentées d'1 bit dès que l'on atteint une nouvelle puissance de 2.

Dans notre exemple lorsqu'on obtient une adresse à écrire de taille plus grande que 255, les adresses sont incrémentées d'1 bit. Plusieurs méthodes sont possibles pour permettre de prévenir le décompresseur du changement de la taille des adresses.

Premièrement, l'utilisation d'un code spécial est nécessaire qui a pour adresse par exemple 255 et signifie que les adresses seront incrémentées de 1 bits à partir de sa lecture. Ainsi le décompresseur est prévenu lors de la lecture d'un telle code et sait alors que le nombre de bits à lire est incrémenté (dans ce cas ci soit 9 bits). Ce processus fonctionne si l'on condamne certaines adresses du dictionnaire (255, 511,1023 ...,) en n'y plaçant pas de séquences d'octets.

Remarque : Un problème se pose, c'est la façon dont on peut différencier la vraie valeur '255' de l'indicateur d'augmentation de bits. Il suffit pour résoudre ce problème d'émettre l'adresse 255 sur 8 bits suivi immédiatement du code '255' mais sur 9 bits soit 011111111.

Une autre méthode, moins élégante, qui utilise une entête pour informer l'algorithme de décompression des positions où dans le fichier l'on à augmenter la taille des adresses.

Le code 'SP' peut ne pas être le seul code de communication avec l'algorithme de décompression. D'autres codes peuvent par exemple servir à vider des dictionnaires périodiquement, à créer d'autres dictionnaires : il n'y a pas de limites.

II.3.1.3 Algorithme de décompression :

1) Le principe de l'algorithme LZW de décompression :

L'algorithme de décompression LZW reconstruit le dictionnaire au fur et à mesure de la lecture du fichier compressé. Son processus est très proche de l'algorithme de compression. On commence par lire les codes du fichier en partant d'une taille de 8 bits.

Puis suivant la méthode utilisée, lorsque le décompresseur rencontre un code spécial il augmente la taille de son tampon pour lire les adresses. Il continue à lire les adresses jusqu'à la fin du fichier et il inscrit la séquence correspondant à l'adresse dans le fichier décompresser.

Voici l'algorithme de décompression :

Algorithme : L'algorithme LZW de décompression

Données :

– Le dictionnaire des symboles rencontrés : Dico

– Le fichier à décompresser : Fichier

Résultat :

– Le fichier décompressé : Fichier

Début

a=Séquence (première adresse contenu dans le fichier)
ecrire a
Tant que le fichier n'est pas à sa fin
b=adresse suivante
si (b appartient Dico) s=Séquence(b)
sinon s=concaténation (a, t)
fin si
écrire séquence(s)
t=s [0]
ajouter (Séquence(a)+t) dans Dico
a =b

Fin **Fin tant que**

Exemple d'utilisation de l'algorithme LZW de décompression

A l'aide de l'algorithme LZW, nous voulons décompresser la séquence suivante : « 83.73.256.45.69.84.45.257.257 »

On lit donc la première adresse soit 83. On écrit la séquence associée à cette adresse du dictionnaire. Ensuite on lit l'adresse suivante. On s'assure que l'adresse lue appartient au dictionnaire et que ce n'est pas un code spécial de contrôle, ensuite on écrit la séquence associée à cette adresse. On ajoute dans le dictionnaire la concaténation de l'ancienne adresse lue et la première lettre de la séquence. On continue ensuite avec les adresses suivantes jusqu'à la fin du fichier.

Remarque :

Nous ne nous préoccuperons pas de la taille des adresses inscrites dans le fichier à décompresser.

Voici un tableau résumant les opérations effectuées sur l'exemple lors du déroulement de l'algorithme LZW de décompression :

	Phase 1	Phase 2	Phase 3	Phase 4
Valeur de a	S=séquence(83)	I=séquence(73)	SI=séquence(256)	-=séquence(45)
Valeur de b	73	256	45	69
b appartient au Dico ?	Oui	Oui	Oui	Oui
Valeur de s	SI=séquence(73)	SI=séquence(256)	- =séquence(45)	E=séquence(69)
Valeur de t	I	SI	-	E
Ajout dans le Dico	SI	ISI	SI-	-E
Ecrire séquence	S et I	SI	-	E

	Phase 5	Phase 6	Phase 7	Phase 8
Valeur de a	E=séquence(69)	T=séquence(84)	-=séquence(45)	IS=séquence(257)
Valeur de b	84	45	257	257
b appartient au Dico ?	Oui	Oui	Oui	Oui
Valeur de s	T=séquence(84)	-=séquence(45)	IS =séquence(257)	IS=séquence(257)
Valeur de t	T	-	IS	IS
Ajout dans le Dico	ET	T-	-IS	ISIS
Ecrire séquence	T	-	IS	IS

Tableau II.4 : Les opérations effectuées lors de déroulement de l'algorithme LZW de décompression

Le dictionnaire généré par l'algorithme de décompression est bien évidemment le même que celui généré par l'algorithme de compression. (Le dictionnaire produit précédemment par l'algorithme de compression.) Nous retrouvons donc notre phrase de départ ('SISI-ET-ISIS'), et nous pouvons aussi remarquer que l'algorithme de décompression est nettement plus rapide que l'algorithme de compression. Dans notre exemple seulement 8 phases ont été nécessaires pour décompresser le fichier alors que la compression a en nécessité 12. L'algorithme de décompression est en moyenne 3 à 10 fois plus rapide que celui de la compression.

II.3.1.4 Efficacité de l'algorithme LZW :

La phrase « SISI-ET-ISIS » est codée sur 96 bits (12*8 bits (dans le code ASCII)).
Après utilisation de l'algorithme LZW de compression, le fichier compressé possède une taille de 79 bits (2*8+7*9), donc on a ici un taux de compression de l'ordre de

$$P = 1 - \frac{79}{96} = \mathbf{0.18} = \mathbf{18}\%$$

En général lorsque la taille du fichier n'est pas trop petite, le taux de compression est supérieur à celui obtenu par une méthode statistique. On peut rajouter que l'algorithme LZW est nettement plus rapide qu'un algorithme de type statistique en terme de traitement.
On peut noter qu'il est possible de compresser un fichier déjà compressé auparavant par un algorithme de type statistique ou l'inverse.

- L'algorithme LZW est aujourd'hui considéré comme la méthode de compression la plus efficace et une des plus connues. Elle est relativement rapide ce qui a rendu l'utilisation de la compression possible sur les disques durs de façon transparente. Cette méthode est aussi utilisée dans le format .GIF, mais encore dans les compresseurs tels que ZIP.

II.4 Codage par automate à états finis : [18]

Les automates à nombre fini d'états sont utilisés entre autres pour le stockage des dictionnaires de langues. L'atout majeur de cette représentation est l'accès rapide à l'information. Cependant, le volume de tels automates reste imposant. Ainsi, comme dans un texte, quand la quantité d'information est suffisamment importante, il devient judicieux de la compresser. On va décrire brièvement comment compresser des automates représentant des

dictionnaires. Ainsi, nous donnons un exemple qui permet de réduire l'espace mémoire nécessaire au stockage des automates.

Nous nous sommes intéressés à deux types de sous-automate série et parallèle et à leurs compression, ainsi l'automate des suffixes (Directed Acyclic Word Graph « DAWG ») permet d'avoir accès, non seulement aux positions des séries et des parallèles dans l'automate, mais aussi, à l'ensemble des positions des sous-séries et des sous-parallèles dans ce même automate.

II.4.1 Principe :

Le codage par automate consiste à représenter un texte sous forme d'un arbre (automate)

Exemple :

La figure suivante montre la représentation des mots suivants :

« Chat chien »

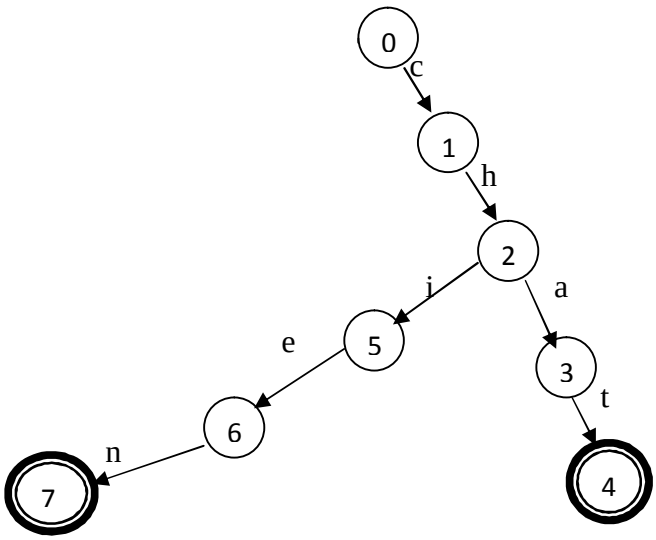


Figure II.6 : Exemple codage par automate à états finis

II.4.2 Définition, notation et abréviation :

II.4.2.1 Automates et sous-automates :

Soit **A=< Σ, Q, δ, q_i, q_f >** un automate déterministe minimal acyclique à nombre fini d'états où : **Σ** est l'alphabet

Q est l'ensemble fini d'états

δ est une fonction de transition **δ : Q × Σ→Q**

q_i est l'état initial et q_f est l'état final

Les ensembles de successeurs et de prédécesseurs d'un état p Є Q sont toujours calculés dans l'automate A comme suit :

Succ(p) = {q Є Q : □ α Є Σ : δ (p, α) =q} et Succ*(p) = {q Є Q : □w Є Σ* : δ (p, w)=q}

Pred(p) = {q Є Q : □ α Є Σ : δ (q, α) =p} et Pred*(p) = {q Є Q : □w Є Σ* : δ (q, w)=p}

Succ(p) (resp. Pred(p)) est l'ensemble des successeurs (resp. prédécesseurs) immédiats de p. Succ*(p) (resp. Pred*(p)) est l'ensemble de tous les successeurs (resp. prédécesseurs) de p, incluant l'état p. Étant donné que nous souhaitons rechercher des structures isolées qui peuvent être extraites de l'automate et remplacées par une transition, nous avons retenu la définition suivante d'un sous-automate **[19].**

Soit A=< Σ, Q, δ, q_i, q_f > un automate acyclique à nombre fini d'états, A' =< Σ, Q', δ', s_i, s_f > est un sous-automate de A si et seulement si :

-Q' ⊂ Q, {s_i, s_f}⊂ Q',

-δ' : $Q' \times \Sigma \rightarrow Q'$

$\forall (q,\alpha) \in Q' \setminus \{s_i, s_f\} \times \Sigma$: Si δ (q, α) est défini alors δ' (q, α)= δ (q, α) sinon δ' (q, α) est indéfini

□ $\forall \alpha \Sigma$, si δ (s_i, α) Є Q' alors δ' (q, α)= δ (q, α) sinon δ' (q, α) est indéfini

$\forall \alpha \Sigma$, si δ' (s_f, α) est indéfini

-∀ q Є Q', q Є Succ*(s_i) et q Є Pred*(s_f)

- ∀q Є Q'\ {s_i, sf} : Succ(q) ⊂ Q' et Pred(q) ⊂ Q'.

La recherche des sous-automates a révélé la présence massive de sous-automates de type série ou parallèle, raison pour laquelle nous nous intéressons spécifiquement à ces deux types de sous-structures et à leur compression.

II.4.2.2 Représentation et stockage des automates :

Le travail s'applique à des automates acycliques avec un seul état final **[20].**

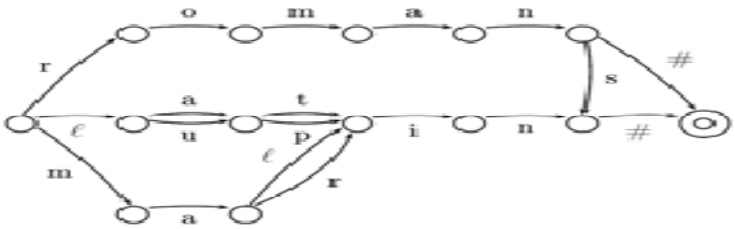
En mémoire, nous représentons l'automate par la liste des transitions sortantes de chaque état. Chacune de ces transitions porte trois informations : un booléen mis à zéro ou à 1 selon que la transition partage ou pas le même état source que celle qui la précède dans la liste, son étiquette et l'adresse de la première transition de son état but (Exemple donné **figure II.6**).

II.4.2.3 Représentation étendu et stockage d'un automate :

En s'appuyant sur la méthode présentée dans **[21**], on peut remplacer itérativement les sous-automates séries et les sous-automates parallèles par de simples transitions portant un nouveau label jusqu'à ce que l'automate en soit dénué. Cela revient en fait à étendre l'alphabet avec les nouveaux labels (voir **figure II.6**). Pour cela on numérote à partir de 1 l'alphabet initial et on associe les numéros au delà de la dernière lettre aux sous-structures traitées. Le stockage se décompose en deux parties, le fichier habituel décrivant un automate auquel s'ajoute un fichier répertoriant l'alphabet étendu. Une série est représentée par la concaténation des étiquettes de ses transitions dans leur ordre d'apparition. Un parallèle est lui aussi représenté par la concaténation des étiquettes de ses transitions selon un ordre préétabli sur l'alphabet. De plus, il est associé à chaque mot un booléen précisant la nature de la sous-structure qu'il représente, il prend la valeur vrai pour désigner une série et faux pour désigner un parallèle, ce qui est noté respectivement S et P sur la **figure II.6**

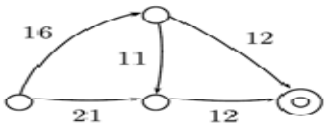
Automate initial A

1	1	ℓ	7
2	0	m	6
3	0	r	4
4	1	o	5
5	1	m	9
6	1	a	10
7	1	a	12
8	0	u	12
9	1	a	14
10	1	ℓ	15
11	0	r	15
12	1	p	15
13	0	t	15
14	1	n	17
15	1	i	16
16	1	n	19
17	1	#	0
18	0	s	19
19	1	#	0



Représentation étendue de A

1	1	16	3
2	0	21	5
3	1	11	5
4	0	12	0
5	1	12	0



Alphabet		
Alphabet initial		
1		ℓ
2		m
3		r
4		o
5		a
6		u
7		p
8		t
9		n
10		i
11		s
12		#
Alphabet étendu		
13	P	5 6
14	P	7 8
15	P	1 3
16	S	3 4 2 5 9
17	S	1 13 14
18	S	2 5 15
19	S	10 9
20	P	17 18
21	S	20 19

Figure II.7: Représentation étendu d'un automate

II.4.2.4 Optimisation de l'occupation mémoire d'un automate :

Les automates sont souvent stockés dans des fichiers textes. Dans le but de réduire l'espace mémoire nécessaire, on a codé chaque donnée du même type sur un nombre de bits minimal, mais constant. Cette méthode permet, d'une part, de stocker les automates à moindre coût et, d'autre part, de conserver un accès efficace aux données. Le codage réservé à chaque donnée est le suivant :

1. Codage du booléen : 1 bit.

2. Codage des étiquettes : le nombre de bits nécessaires pour coder l'alphabet, noté

Taille alphabet : **Taille alphabet = [log2 (|∑'|)]** où ∑' est l'union de l'alphabet initial ∑ et des nouveaux caractères alloués aux séries et parallèles détectés et factorisés.

3. Codage de l'adresse de l'état d'arrivée : le nombre de bits nécessaires pour coder les adresses, noté Taille adresse : **Taille adresse= [log2 (ValMax+1)]** où ValMax, correspondant à l'adresse de l'état but de valeur maximale.

L'adresse est une adresse absolue puisque le dictionnaire contient des mots courts, ce qui nécessite des grands sauts d'adresse et qui rend donc inutile un adressage relatif.

Exemple 1 : La figure II.6 présente un automate initial et sa représentation en mémoire ainsi que sa représentation étendue où il est composé de 4 états et 5 transitions. À cet automate est associé un alphabet étendu contenant l'alphabet initial, 12 caractères et 9 sous-structures distribuées en 4 parallèles et 5 séries. Ainsi, le « caractère » 13 représente le parallèle composé des transitions a et u. Dans cet exemple, 10 bits sont nécessaires pour coder l'automate initial car : **Taille alphabet =4 et Taille adresse =5.**

II.5 Conclusion :

Nous avons présenté dans ce chapitre les différentes techniques de compression de données qui sont extrêmement diverses et chacun d'entre elle tente d'être le plus efficace que possible.ces méthodes que nous avons présentées sont des techniques très efficaces et très utilisées dans différents domaines d'application et elles sont à l'origine de plusieurs utilitaires de compression (Win Zip, Win RAR…) utilisés dans tous les systèmes informatique. Nous remarquons que chaque famille à son principe de compression déférent d'une autre. Dans la mesure ou l'une se base sur un concept statique, permettant de recenser le nombre d'apparition, et l'autre sur le concept de mémorisation (dictionnaire) qui permet de retenir de coder la plus longue chaine de caractère qui existe dans un message .on a aussi présenté d'autres techniques par automate à états finis qui sont utilisés entre autres pour le stockage des dictionnaires de langues. En fin ce domaine est en progression car de nouvelles méthodes sont découvertes dans le but d'améliorer le taux et le temps de compression.

Chapitre III :
Analyse et conception

III.1 Introduction :

A la suite des deux chapitres précédents dans lesquels on a abordés les généralités sur la compression de données et les méthodes de compression de données nous allons présenter dans la première partie la nouvelle méthode de compression proposée qui consiste à compresser les données à l'aide d'automate d'états finis déterministe dans le but d'intégrer la compression de données.

Nous consacrons la deuxième partie pour la conception dont nous présentons les différents modules de notre application qui fait la compression ainsi nous expliquons les deux processus de codage et de décodage.

III.2 Analyse de la méthode proposée:

Dans cette partie nous expliquons la nouvelle méthode de compression de données qui est classé parmi les méthodes sans perte car après la décompression on peut avoir exactement le fichier original.

Cette technique est basée sur les automates d'états finis déterministes et consiste à représenter une structure efficace pour la compression et la décompression de données.

III.2.1 Principe de la méthode proposée :

III.2.1.1 Le codage :

L'algorithme de construction repose sur une lecture du fichier et consiste à :

- Analyse lexicale du fichier (scanner).
- Construire au fur et à mesure l'automate d'états finis déterministe.
- Transformer l'automate sous forme d'une matrice.
- Transformer la matrice sous forme d'un arbre linéaire.
- Obtenir une nouvelle matrice à partir de l'arbre.
- Compression de la matrice.
- Sauvegarder les résultats.

Voici le schéma général de codage :

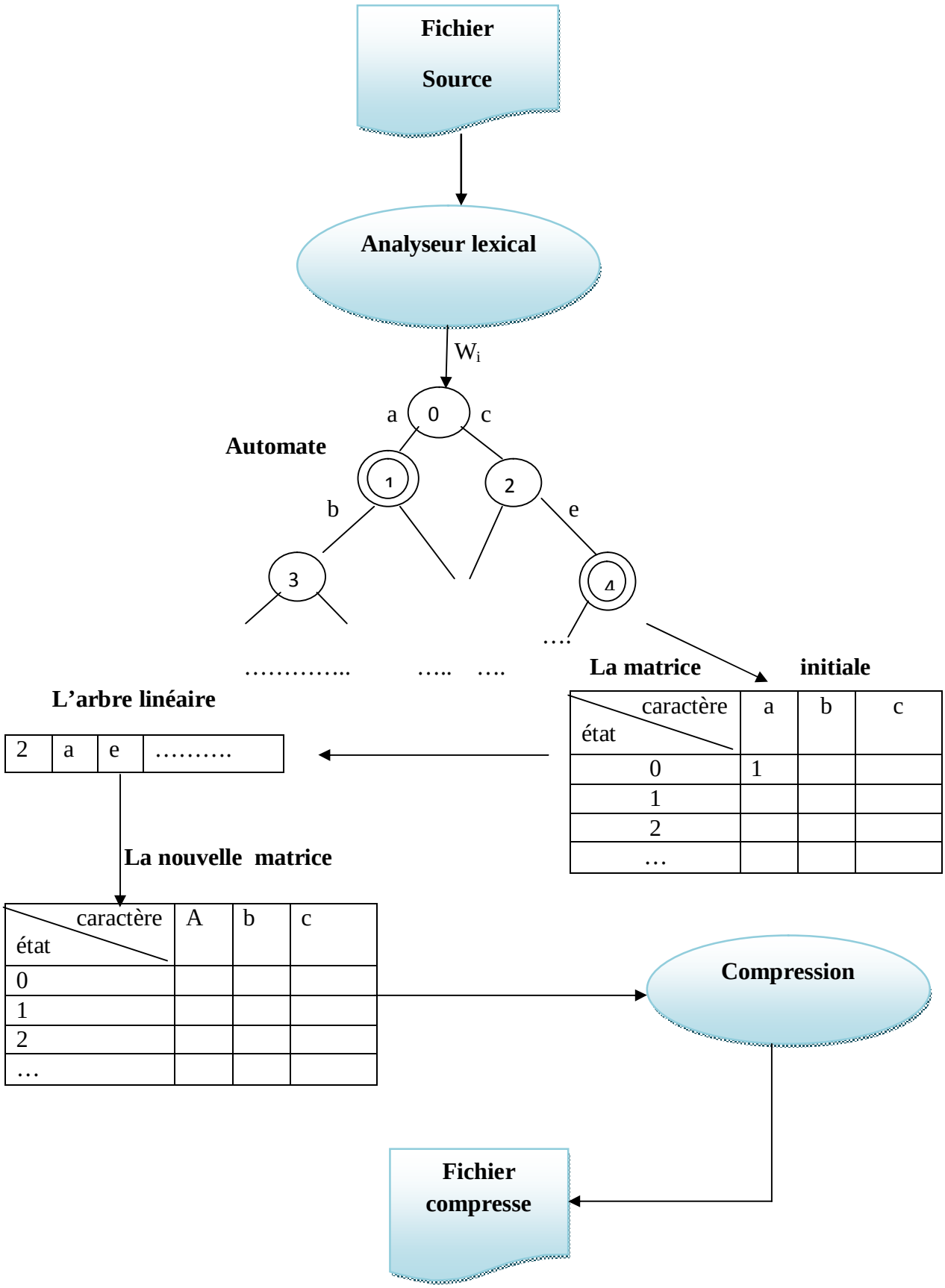


Figure III.1 : schéma général de codage

III.2.1.1.1 Exemple de codage:

Nous allons effectuer la compression de notre message : « ba a ab bb »

On suit les étapes ci-dessous :

1)-Construction de l'automate :

Voici l'automate qui reconnait les mots définit sur l'alphabet {a, sp, b}, sp : l'espace

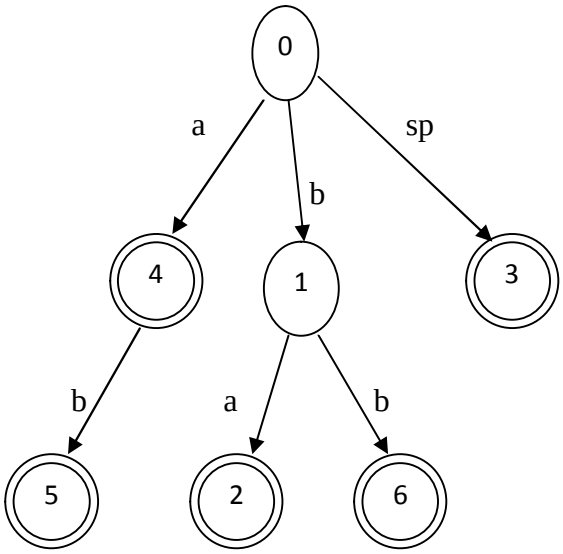


Figure III.2: Automate représentant « ba a ab bb »

2)-Transformation sous forme d'une matrice :

La transition de l'état initial « 0 » vers l'état « 1 » est régie par un b, ainsi on peut passer de l'état « 0 » par un espace vers l'état « 3 », aussi par un a vers l'état « 4 », on passe de l'état 4 par un b vers l'état « 5 » qui est un état final...etc. s'il n' ya pas de transition on met« - »

La table de transition associée à cet automate est alors :

Etats	NB	sp	a	b
0	3	3	4	1
1	2	-	2	6
2	0	-	-	-
3	0	-	-	-
4	1	-	-	5
5	0	-	-	-
6	0	-	-	-

NB : Le nombre de transitions pour chaque état.

- La matrice associée est :

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

Figure III.3: La matrice associée à l'automate

3)-Transformation sous forme d'un arbre linéaire :

Nous allons transformer la matrice précédente à un arbre linéaire suivant cette démarche :

Nous parcourons l'automate niveau par niveau, pour le premier niveau l'état initial « 0 » a trois éléments non nuls qui sont 3, 4, et 1 leurs indices sont respectivement l'espace, a, et b

~~0~~	3	4	1

De cette façon on obtient L'arbre linéaire pour le premier niveau qui est 3 sp a b.

Pour le deuxième niveau, on commence par l'état « 3 » qu'est un état final, il n'a pas de transitions. On passe à l'état « 4 » qui a un seul élément non nul qui est 5, pour l'état « 1 » il a deux éléments non nuls qui sont 2, et 6.

0	~~3~~	4	1	0

0	3	~~4~~	1	0	5

0	3	4	~~1~~	0	5	2	6

L'arbre linéaire pour le deuxième niveau est 0 1 b 2 a b.

Pour le dernier niveau on remarque que les états sont tous des états finaux donc il n' ya pas de transitions.

0	3	4	~~1~~	0	5	2	6	0	0	0

Enfin on obtient l'arbre linéaire de toute la matrice qui est : 3 sp a b 0 1 b 2 a b 0 0 0

Après cette étape on construit une nouvelle matrice associée à l'arbre linéaire.

- **La table de transition :**

états	NB	sp	A	B
0	3	1	2	3
1	3	-	-	-
2	1	-	-	4
3	4	-	5	6
4	0	-	-	-
5	0	-	-	-
6	0	-	-	-

- **La matrice associée est :**

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

Figure III.4 : La matrice associée à l'arbre

Si on construit l'automate à partir de cette matrice on obtient exactement le même que le précédent (**Figure III.4**) :

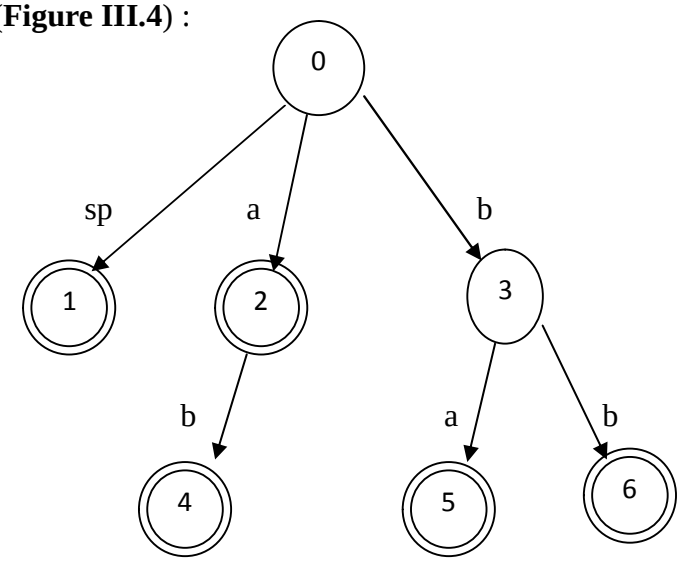


Figure III.5: Automate obtenu à partir de la matrice

Le résultat de la compression sa sera les états finaux de tous les mots.

Pour l'exemple « ba a ab bb », l'état final de ba est 5, l'état final pour sp est 1, pour a c'est 2…etc. Donc le résultat de la compression est :

5	1	2	1	4	1	6

III.2.2 Le décodage :

Après avoir le fichier compressé, nous allons passer à la décompression du ce fichier pour reconstitué le fichier original.

La décompression c'est l'inverse de la compression, on construit tout d'abord la matrice à partir de l'arbre puis on peut reconstituer le fichier source après plusieurs étapes qu'on détaillera dans l'algorithme de décompression.

Voici le schéma général de décodage :

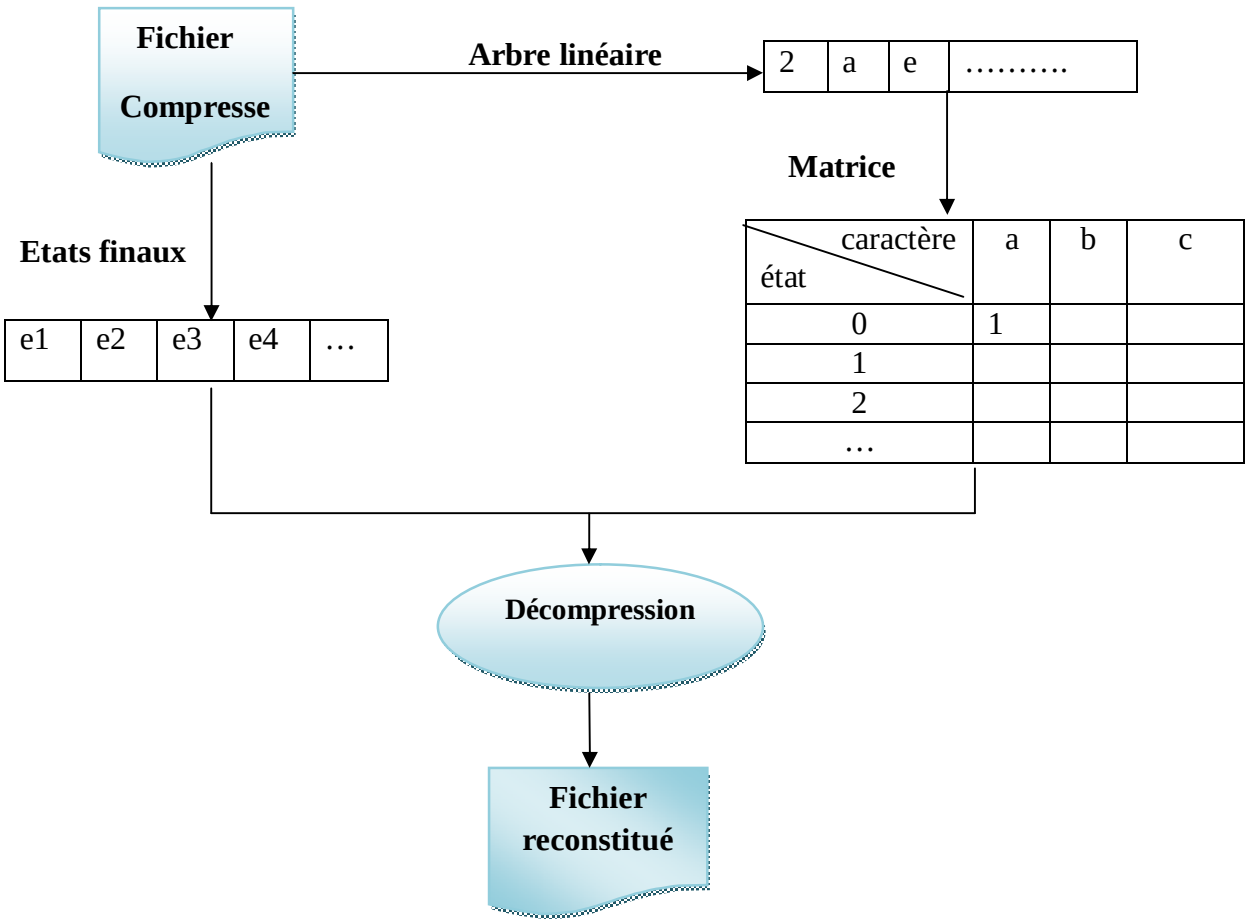


Figure III.6: schéma général de décodage

Pour l'exemple que nous avons donné précédemment pour la compression on a obtenu le résultat suivant:

5	1	2	1	4	1	6

Après la décompression on aura le fichier reconstitué qui est le fichier original sans aucun changement (compression sans perte) ci-dessous :

ba a ab bb

III.3 Conception :

Une bonne conception est facile à réaliser, à maintenir et à comprendre bien qu'elle doit fonctionner correctement.

III.3.1 Classement des méthodes de conception :

III.3.1.1 Conception fonctionnelle descendante (top-down) :

Pour ce type de méthode, le système est conçu d'un point de vue fonctionnel en commençant au niveau le plus général et en descendant progressivement vers la conception détaillée.

III.3.1.2 Conception fonctionnelle ascendante (bottom-up) :

Est le contraire de la conception descendante, dans celle-ci en commence par les sous composants et puis en remonte peu à peu vers le niveau le plus général.

III.3.2 Quelques critères de qualité:

Il ya cinq (05) critères de qualité à respecter lors de la définition des modules :

1) **La décomposition modulaire :** capacité de décomposer un problème en sous-problèmes. Semblable à la méthode descendante.
2) **La composition modulaire :** capacité de recombinaison de modules écrits, semblable à la programmation ascendante.
3) **La continuité modulaire :** capacité à réduire l'impact de changements dans les spécifications à un minimum de modules liés entre eux, et mieux à un seul module.
4) **La compréhension modulaire :** capacité à l'interprétation par un programmeur du fonctionnement d'un module ou d'un ensemble de modules liés, sans avoir à connaitre tout le logiciel.

5) **La protection modulaire :** capacité à limiter les effets produits par des incidents lors de l'exécution à un nombre minimal de modules liés entre eux, mieux à un seul module.

III.3.3 Conception de notre travail :

Dans notre application nous allons donner une grande importance au concept de module qui est considéré comme les taches essentielles réalisées par l'application.la visualisation de ces taches donnent un aperçu général du rôle de l'application. Cette décomposition du problème en modules va nous faciliter le travail.

Le rôle de notre application est de réaliser la nouvelle méthode compression/décompression de données proposée afin de réduire la taille des données pour gagner d'espace mémoire et le temps de traitement.

D'après la description donnée au dessus nous allons adopter la méthode descendante pour réaliser notre application qui consiste à décomposer le problème en sous problèmes.

III.3.4 Les différents modules qui composent notre application :

Pour aboutir à l'objectif de notre application, cette dernière doit se décomposée en quatre modules principaux suivants :

1-Module gestion des fichiers

2-Module de compression

3-Module de décompression

4-Module calcul et affichage des performances

L'application doit prendre en charge les entrées et les sorties par le premier module (gestion des fichiers), celui-ci sera sollicité par le deuxième ou le troisième module (compression ou décompression) pour compléter la tache choisis par l'utilisateur, le calcule et affichage des performances sera fait au fur et à mesure dans le quatrième module (performances).

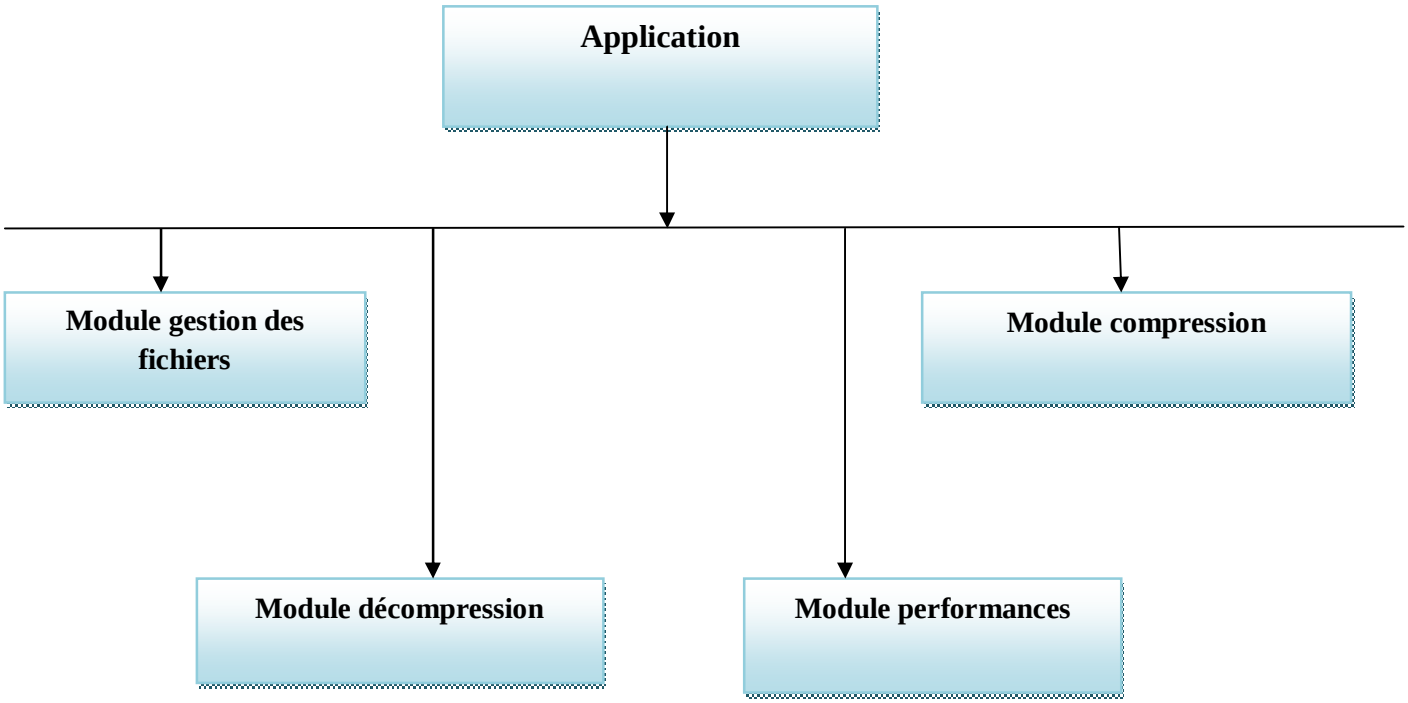


Figure III.7: Module globale de l'application

III.3.4.1 Le module de gestion de fichiers :

Ce module nous permet de faire la gestion des fichiers manipulés par les autres modules compression, décompression et performances.

- Il ouvre un fichier source en lecture qui porte l'extension (.txt) pour le compresser.
- Il, ouvre un nouveau fichier en écriture qui porte l'extension (.Au), ce fichier contiendra le résultat de compression.
- Il ouvre un nouveau fichier en écriture qui contiendra le résultat de décompression.

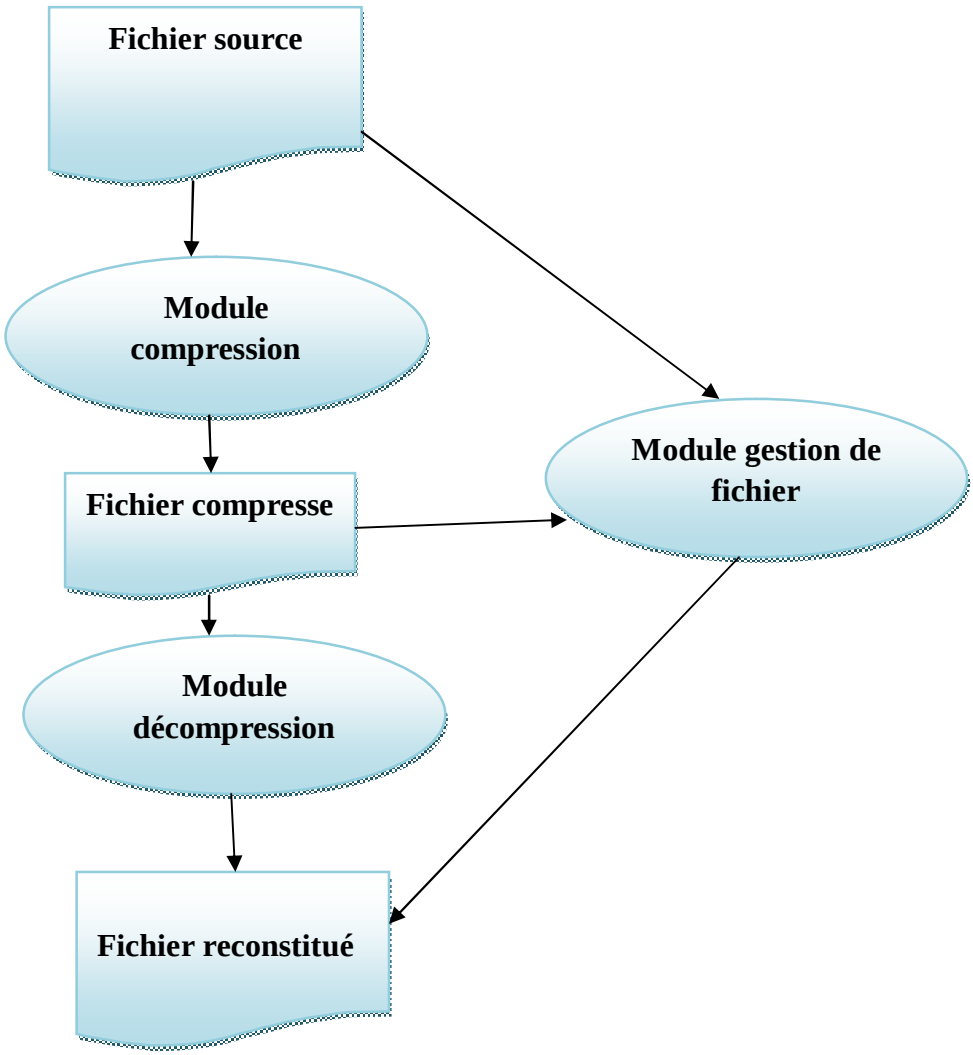


Figure III.8: Gestion des fichiers traités par le module compression et décompression

III.3.4.2 Le module de compression :

Ce module peut être considérer comme étant la partie « action » du module principal de l'application.il se charge de la compression de fichiers.

- Le module de gestion des fichiers charge le fichier élu par l'utilisateur.
- La fonction FileOpen() va ouvrir le fichier source et calcule la taille de ce fichier ainsi le nombre de caractères.
- Une fonction analyselexical() qui retourne tous les tokens (mots) que le fichier contient.
- La fonction Automate () qui calcule l'état courant et l'état suivant de chaque caractère et le nombre d'états, puis elle construit la matrice initiale qui va contenir l'état suivant de chaque caractère.
- Une fonction Arbre () qui construit l'arbre linéaire de la matrice calculée précédemment.

- Une fonction Matrice () qui construit la nouvelle matrice à partir de l'arbre.
- Une fonction TableCompress() qui va compresser la matrice.

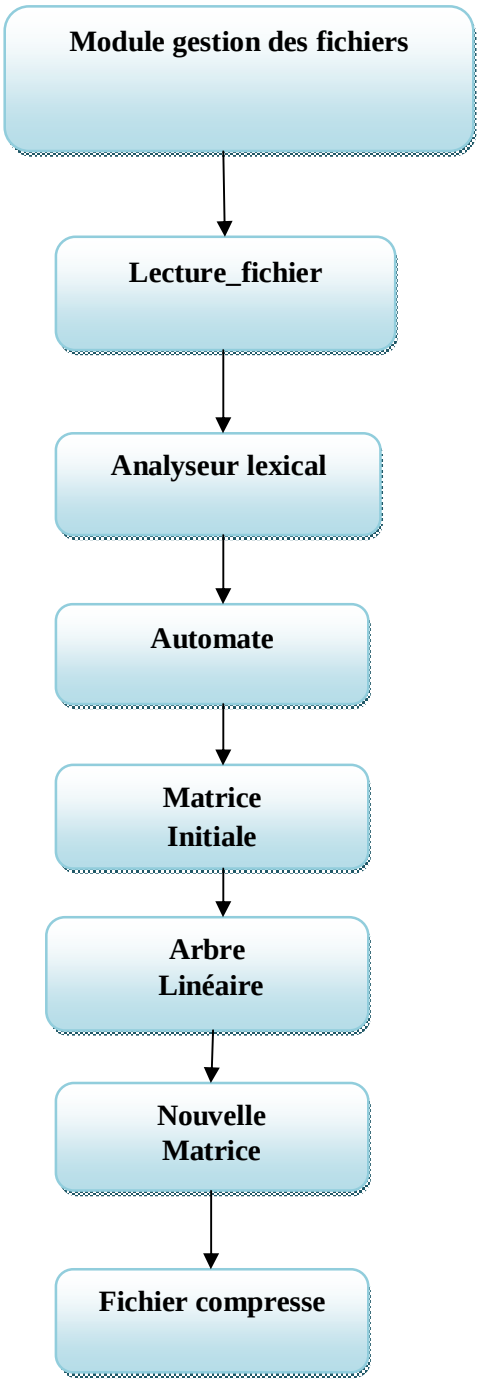


Figure III.9 : Module de compression

III.3.4.3 Le module de décompression :

Le module de gestion de fichier charge le fichier élu par l'utilisateur.

Si le fichier est un fichier compressé alors :

Une fonction Lire_Fichier() qui va lire le fichier compressé

Une fonction Ecrire_Fichier() qui va écrire le résultat de la décompression dans un fichier ouvert en écriture par le gestionnaire de fichier, qu'on appel le « fichier reconstitué ».

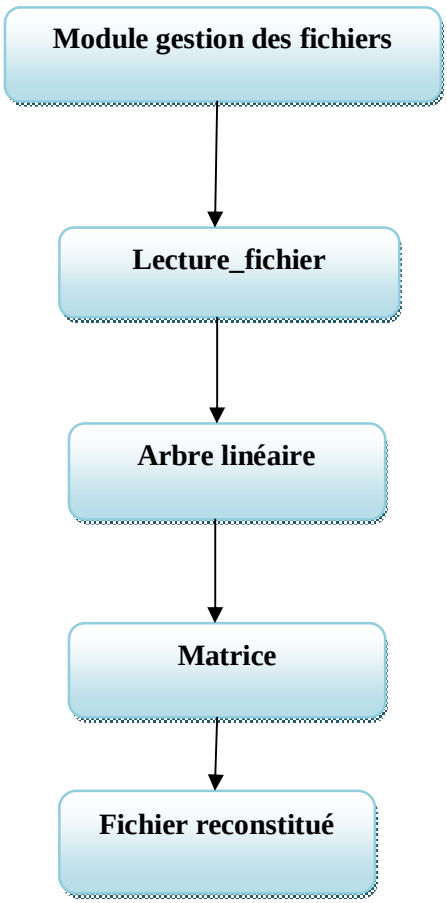


Figure III.10 : Module de décompression

III.3.4.4 Module affichage et calcul des performances :

Lors de la compression et la décompression il ya un module qui est charger de calculer et afficher les performances de l'algorithme comme la taille du fichier source (Taille_Src()), la taille du fichier compressé (Taille_ Comp), et le taux de compression (Taux_Comp()).

C’est grâce à celui-ci que nous allons tester et évaluer cette nouvelle méthode de compression.

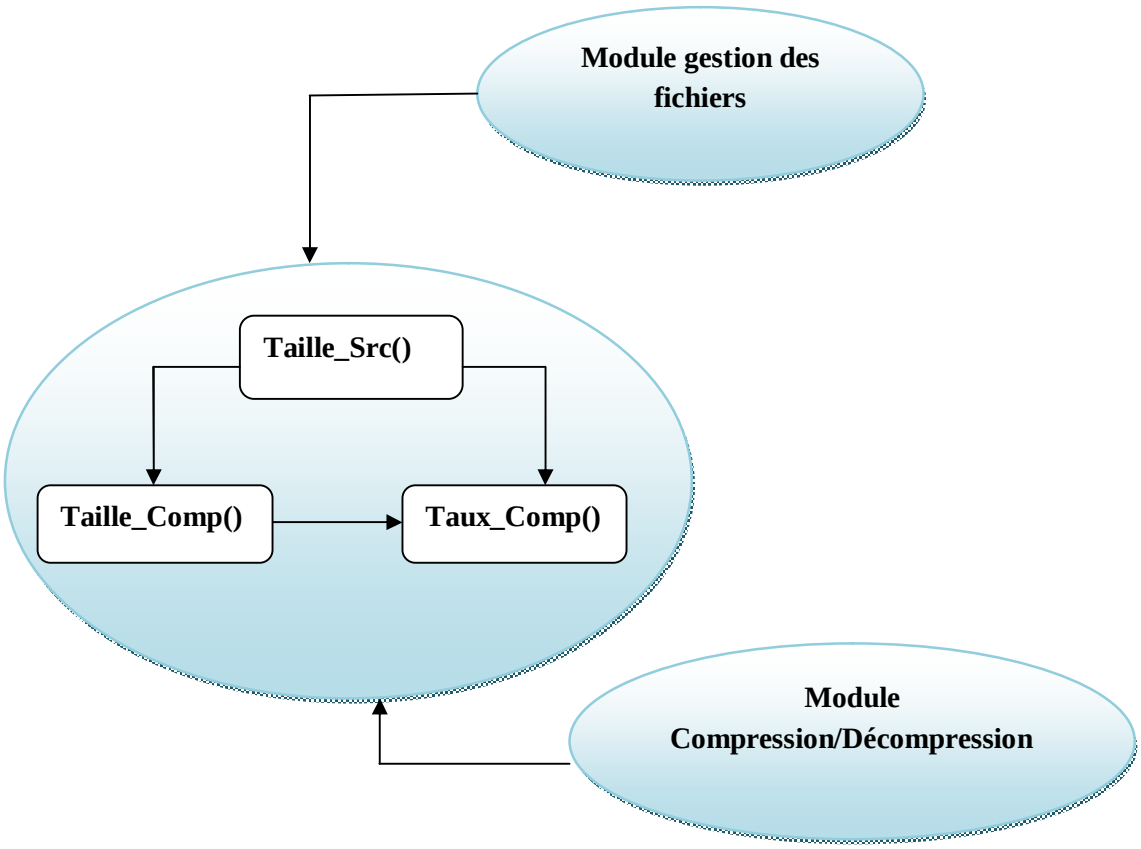


Figure III.11: Module calcul et affichage des performances

III.4 Les algorithmes de chaque module :

III.4.1 Algorithme du module de gestion de fichiers :

Ce module possède deux fonctions qui gèrent les entrées/sorties de notre application, une fonction Lecture_Fichier pour ouvrir des fichiers en lecture, et une fonction Ecriture_Fichier pour ouvrir des fichiers en écriture.

Lecture_Fichier :

Début

Si (Compression)

1. charger le nom du fichier, son extension, et son emplacement.
2. vérifier : **Si** (fichier existe) alors
   - Ø ouverture de fichier source en lecture.
   - Ø lecture de fichier source mot par mot et Les envoyés au module compression.
   - Ø fermeture de fichier source.

Finsi **Finsi**

Sinon

Si (Décompression)

1. récupérer le nom du fichier à décompressé, son extension et son emplacement.
2. vérifier **Si** (fichier n'est pas vide) alors
   - Ø ouverture de fichier compressé en lecture.
   - Ø lecture de fichier compressé au fur et à mesure et les envoyés au module décompression
   - Ø fermeture de fichier compressé.

Finsi

Finsi

Fin

Ecriture_Fichier:

Début

Si (Compression)

- Ø crier un nouveau fichier à qui on donne le nom du fichier source qui porte l'extension « .txt».
- Ø ouvrir le fichier crie en écriture.
- Ø écrire dans le fichier les codes des caractères.
- Ø fermeture de fichier compressé.

Sinon

Si (Décompression)

- Ø crier un nouveau fichier à qui on donne le nom de fichier compressé qui porte L'extension « .Au ».
- Ø ouvrir le fichier reconstitué en écriture.
- Ø écrire son contenu caractère par caractère.
- Ø fermeture de fichier décompressé.

Finsi

Fin

III.4.2 Algorithme du module de compression :

Début

- Ouvrir le fichier à compresser en lecture

-créer un nouveau fichier avec l'extension « .Au » qui contiendra le résultat de la compression.

Tant qu'on n'est pas à la fin du fichier **faire**

- lire le fichier original mot par mot via l'analyseur lexical

-construire l'automate de touts les mots du fichier source.

-représenter l'automate par une matrice à deux dimensions (états\caractère)

Par :

Tant que ce n'est pas la fin du mot **faire**

- l lire le mot caractère par caractère.

Si Tableau [état courant] [caractère] reçoit zéro **alors**

- mettre état suivant dans le tableau à deux démentions.
- incrémenté l'état courant.

Sinon

- incrémenté l'état courant.

Finsi

FinTantque

FinTantque

// Construction de l'arbre linéaire

Tant que la position de l'état courant inférieure à la position de l'état final **faire**

- Parcourir l'automate niveau par niveau.
- calculer l'occurrence des caractères de chaque niveau de l'automate.
- mettre l'occurrence dans l'arbre.

-initialiser l'occurrence à zéro.

- incrémenter la position d'arbre.

Si le tableau n'est pas vide **alors**

-mettre le caractère dans l'arbre.

-incrémenté la position de l'arbre.

- incrémenté la position de l'état final.

Finsi

-incrémenté la position de l'état courant.

FinTantque

//construction de la nouvelle matrice

- initialiser le tableau à zéro

-initialiser le nombre d'états a un.

Tant que se n'est pas la fin de l'arbre **faire**

- Lire l'arbre caractère par caractère
- Tableau de caractère reçoit le nombre d'états.
- incrémenté nombre d'états.
- incrémenté la position de l'arbre.

FinTantque

//Table de compression
Pour i allons de 0 jusqu'à la taille du mot **faire**
Retourner l'état final de chaque mot c'est-à-dire
Etat_final reçoit mat [etat_final][caractère]
Fin pour
End.

III.4.3 Algorithme du module de décompression :

Début

- Ø ouvrir le fichier à décompresser en lecture
- Ø créer un nouveau fichier qui contiendra le résultat de la décompression
- Ø lire l'entête de fichier à décompresser
- Ø ouvrir le nouveau fichier en écriture

Tant que ce n'est pas la fin de l'entête (arbre) **faire**

- Reconstruire l'automate (matrice) à partir de l'arbre enregistré dans l'entête.
- Décrémenter la Taille de fichier.

Fin tant que

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

- Récupérer le mot correspond à l'état final enregistré dans le fichier.
- On écrit le mot dans le nouveau fichier ouvert en écriture.

Fin tant que

- Ø enregistrer le nouveau fichier

Fin

III.4.4 Algorithme du module de calcul et affichage des performances:

- **Calcule la taille du fichier :**

Début

-Ouvrir le fichier en lecture.

-Positionner le curseur à la fin de fichier.

-récupérer la position du curseur.

-Taille = position du curseur

Fin

- **Calcule du taux de compression :**

Début

-récupérer la taille de fichier original.

-récupérer la taille de fichier compressé.

-Taux = ((la taille_orig-taille_comp)*100/taille_orig)

Fin

III.5 Conclusion :

Dans ce chapitre nous avons présenté la partie analyse et conception de notre projet. Nous avons fait une conception détaillée pour les différents modules qui compose notre application, et nous avons donné l'algorithme de chaque module. Dans le chapitre qui suit nous allons faire l'implémentation et l'évaluation de notre méthode en se basant sur cette conception.

Chapitre IV : Implémentation et évaluation

IV.1 Introduction :

Dans ce chapitre nous allons présenter dans une première partie l'implémentation de notre travail ; dans la deuxième partie nous allons effectuer des tests qui sont fait afin d'évaluer cette nouvelle méthode, ces tests sont fait à base de plusieurs critères qui sont :

- Taille du fichier
- Taille du mot
- Taux de compression
- Type de fichier

IV.2 L'environnement technique de développement :

IV.2.1 Présentation du matériel :

Pour réaliser notre application, nous avons utilisé une machine ayant les caractéristiques suivantes :

- Un processeur Intel® Core ™ i3-23330M CPU @2.20 GHZ.
- RAM de 3,00 Go.
- Disque dur de 320 GO.
- Type de système 32 bits.

IV.2.2 Présentation de l'environnement logiciel :

Nous avons travaillé sous un système d'exploitation Microsoft Windows 7 édition familiale premium.

IV.2.2.1 Présentation du langage de programmation utilisé (C++) : [22]

Le c++ est un langage de programmation de haut niveau développé en 1983 Bjarnes Stroustrup aux Laboratoires Bell. Il s'agit d'une version orientée objet de son prédécesseur C. C'est un langage très utilisé pour développer des applications graphiques (comme celles qui tournent sur les environnements Windows et Macintosh).

IV.2.2.2 Les principales raisons du succès de C++ : [22]

Ce choix est motivé par diverses raisons, car le langage C++ reste l'un des langages les plus utilisés actuellement.

- Le **C++** permet la programmation sous de multiples paradigmes comme la programmation procédurale, la programmation orientée objet et la programmation générique .
- C + + est l'un des langages de programmation les plus populaires avec une grande variété de plateformes matérielles et systèmes d'exploitation.
- La plupart des implémentations des algorithmes sont implémentés à base du langage C++.il est actuellement le 3éme langage le plus utilisé en monde.

IV.2.2.3 Présentation de DEV C++ : [22]

DEV-C++ est un environnement de développement intégré (IDE) permettant de programmer en C et en C++. Développé avec Borland Delphi 6, DEV-C++ était disponible uniquement sous Microsoft Windows. Longtemps à l'abandon, le projet a été repris par un autre développeur en 2011 et il est régulièrement mis à jour.

Dans notre travail on a utilisé la version DEV 4.9, la figure suivante illustre son interface principale.

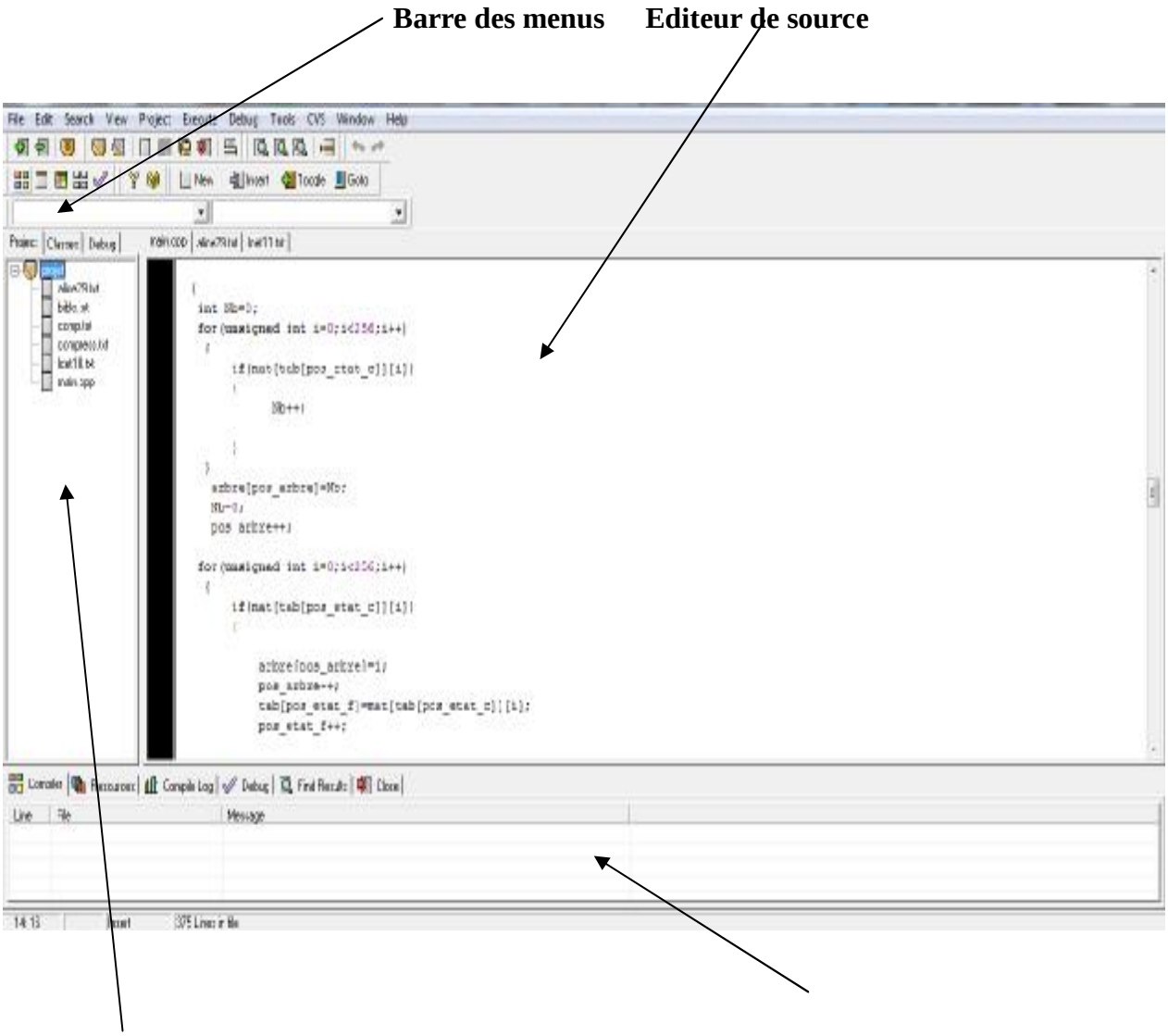


Figure IV.1: L'interface de l'environnement DEV-C++

IV.3 Evaluation :

Pour évaluer le taux de compression nous allons faire des tests sur les fichiers textes, web binaires et images. Ce taux est calculé par la fonction suivante :

$$\textbf{Taux de compression} = \frac{(\boldsymbol{taille\ source - taille\ compresse}) * \mathbf{100}}{\boldsymbol{taille\ source}}$$

Nous allons faire une comparaison entre le taux de compression de ces fichiers par rapport à la taille du mot et au type de fichier. On donne ces valeurs pour la taille du mot : 3, 4, 5, 6, 7.

IV.3.1 Les fichiers textes :

Nous commençons par les fichiers textes, nous allons faire des tests, premièrement avec l'utilisation de l'analyseur lexical (lecture de fichier token par token) et sans analyseur lexical c'est-à-dire avec la variation de la taille du mot (lecture du fichier au fur et à mesure) nous avons pris une collection de données textes (LargeCanterburyCorpus) les résultats obtenus sont représentés dans les tableaux ci-dessous :

1) Avec l'analyseur lexical :

Nom du fichier	Taille du fichier source (ko)	Taille du fichier compressé (ko)	Taux de compression (%)
Lclt10.txt	417	261	12.30
Plrabn1c.txt	471	287	3.16
Bible.txt	3953	1791	14.80
World192.txt	2416	1369	3.07

Tableau IV.1 : Evaluation de taux de compression sur les fichiers textes (avec lex)

- En remarque que le taux de compression est entre 3% et 15%.
- Le taux de compression moyen est de 8.33%.

2) <u>Avec la variation de la taille du mot:</u>

Fichier et sa Taille source	**plrabn12.txt (471 ko)**		**bible.txt (3953 ko)**		**world192.txt (2416 ko)**		**Lcet10.txt (417 ko)**	
Taille du Mot	**Taille Comp (ko)**	**Taux Comp (%)**	**Taille Comp (ko)**	**Taux Comp (%)**	**Taille Comp (ko)**	**Taux Comp (%)**	**Taille Comp (ko)**	**Taux Comp (%)**
3	329	30.18	2653	32.88	1656	31.44	297	28.94
4	287	39.09	2052	48.10	1369	43.34	261	37.52
5	304	35.46	1791	54.70	1308	45.85	271	35.09
6	354	24.97	1753	55.66	1356	43.87	304	27.21
7	413	12.97	1870	52.69	1461	39.54	346	17.00

Tableau IV.2 : Evaluation de taux de compression sur les fichiers textes (sans lex)

-Taille du mot=3

Lorsque la taille du mot égal à **3** le taux de compression varier de 28.94% à 32.88% et le taux de compression moyen est 30.86%.

-Taille du mot =4

Le taux varier de 37.52% à 48.10% et le taux moyen est 42.01%

-Taille du mot=5

Le taux varier de 35.46% à 54.70% et le taux moyen est 42.77%.

-Taille du mot =6

Le taux varier de 24.97% à 55.66% et le taux moyen est 37.92%.

-Taille du mot =7

Le taux varier de 12.97% à 52.69% et le taux moyen est 30.54%.

- De ces résultats on constate que le taux de compression moyen est meilleur lorsque la taille du mot égal à 5.

IV.3.2 Les fichiers web:

Nous avons pris une collection de données (web corpus) pour cette évaluation, comme les fichiers textes nous allons faire des tests avec et sans l'analyseur lexical.

Les résultats obtenus sont représentés dans le **tableau IV.3** ci-dessous :

1) Avec l'analyseur lexical :

Nom du fichier	Taille du fichier source (ko)	Taille du fichier compressé (ko)	Taux de compression (%)
B01.html	2076	2106	-1.05
B02.html	2084	2220	-6.49
B03.html	2123	2158	-1.64

Tableau IV.3 : Evaluation de taux de compression sur les fichiers web (avec lex)

- En remarque que le taux de compression est négatif.

2) Avec la variation de la taille du mot:

Fichier et sa Taille source	B01.html (2076ko)		B02.html (2084ko)		B03.html (2123ko)	
Taille du mot	Taille comp (ko)	Taux Comp (%)	Taille comp (ko)	Taux Comp (%)	Taille comp (ko)	Taux Comp (%)
3	1455	29.95	1452	30.36	1467	30.92
4	1245	40.04	1229	41.04	1218	42.65
5	1233	40.62	1200	42.45	1161	45.34
6	1318	36.53	1262	39.46	1199	43.54
7	1446	30.36	1365	34.51	1293	39.11

Tableau IV.4 : Evaluation de taux de compression sur les fichiers web (sans lex)

-Taille du mot=3

Le taux varier de 29.95% à 31.92% et le taux moyen est 30.41%.

-Taille du mot =4

Le taux varier de 40.04% à 42.65% et le taux moyen est 41.24%

-Taille du mot=5

Le taux varier de 40.45% à 4534% et le taux moyen est 42.80%.

-Taille du mot =6

Le taux varier de 36.53% à 43.54% et le taux moyen est 39.84%

-Taille du mot =7

Le taux varier de 30.36% à 39.11% et le taux moyen est 34.66%

- On constate que le taux de compression moyen est meilleur lorsque la taille du mot égal à **5.**

IV.3.3 Les fichiers binaires :

Pour ce type de fichiers nous allons effectuer des tests sur une collection de données CalgaryCorpus (**tableau IV.5**).

Fichier et sa Taille Source	**Bib (109ko)**		**Book1 (751ko)**		**Book2 (597ko)**		**Pic (502ko)**	
Taille du mot	**Taille comp (ko)**	**Taux Comp (%)**	**Taille comp (ko)**	**Taux Comp (%)**	**Taille comp (ko)**	**Taux Comp (%)**	**Taille Comp (ko)**	**Taux Comp (%)**
3	86	21.30	523	30.41	428	28.38	355	29.17
4	84	23,04	451	39.93	378	36.70	290	42.19
5	90	17.30	471	37.32	390	34.70	260	48.12
6	99	9.40	539	28.23	434	27.35	235	53.28
7	109	0.13	626	16.64	491	17.84	229	54.34

Tableau IV.5 : Evaluation de taux de compression sur les fichiers binaires

-Taille du mot=3

Le taux varier de 21.30% à 30.41% et le taux moyen est 27.31%.

-Taille du mot =4

Le taux varier de 23.04% à 42.19% et le taux moyen est 35.46%

-Taille du mot=5

Le taux varier de 17.30% à 48.12% et le taux moyen est 34.36%.

-Taille du mot =6

Le taux varier de 9.40% à 53.28% et le taux moyen est 29.57%

-Taille du mot =7

Le taux varier de 0.13% à 54.34% et le taux moyen est 22.23%

- On constate que le taux de compression moyen est meilleur lorsque la Taille égal **4.**

IV.3.4 Les données images :

Pour l'évaluation du taux de compression des fichiers images nous avons pris une collection d'image (corpus image) de format tif et dcm car ce format d'images n'est pas compressé contrairement à d'autres formats tels que JPEG, GIF, PNG,…etc.

Les résultats sont représentés dans le tableau suivant :

Fichier Et sa Taille Source	**Lukas_foot_1_t.tif (2184 ko)**		**Lukas_hand_0_tif (2442ko)**		**Lukas_breast_0.dcm (8195 ko)**		**Lukas_breast_1.dcm (8195ko)**	
Taille du mot	**Taille Comp (ko)**	**Taux Comp (%)**	**Taille Comp (ko)**	**Taux Comp (%)**	**Taille Comp (ko)**	**Taux Comp (%)**	**Taille Comp (ko)**	**Taux Comp (%)**
3	1535	29.71	1728	29.24	5513	32.72	5523	32.61
4	1345	38.42	1521	37.70	4202	48.72	5222	48.48
5	1458	33.21	1565	35.90	3712	54.71	3827	53.30

Tableau IV.6 : Evaluation de taux de compression sur les données images

Ø **Taille du mot=3**

Le taux varier de 29.71 % à 32.72% et le taux moyen est 31.07%.

Ø **Taille du mot =4**

Le taux varier de 38.42% à 48.72% et le taux moyen est 43.33%

Ø **Taille du mot=5**

Le taux varier de 33.21% à 54.71% et le taux moyen est 44.28%

- On constate que le taux de compression moyen est meilleur lorsque la Taille égal **5.**

IV.4 Comparaison des résultats selon le type de fichier :

La figure suivante illustre les résultats obtenus :

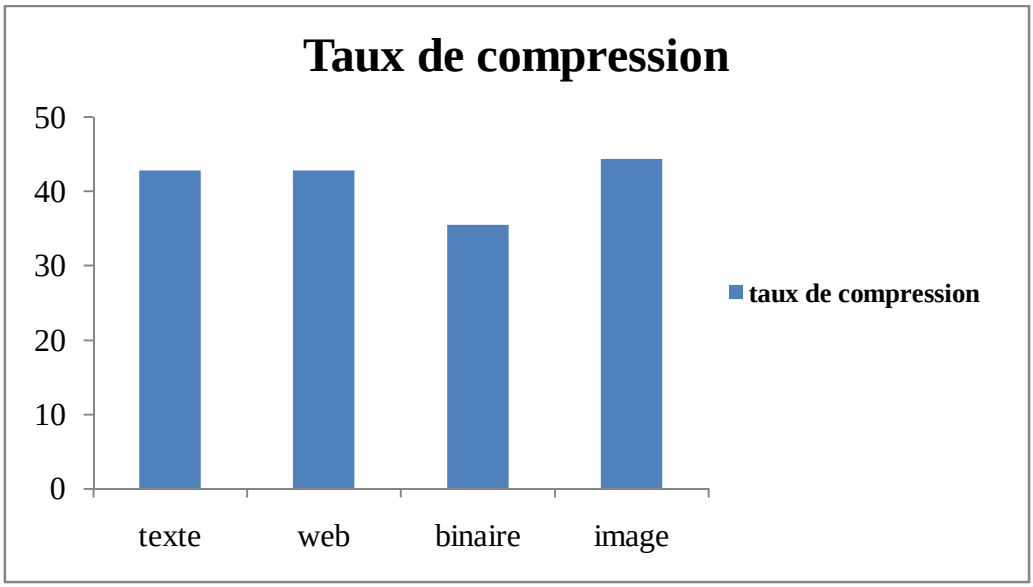


Figure IV.2 : Comparaison de taux de compression selon le type de fichier

Interprétation :

D'après la figure IV.1 on peut constater que le taux de compression est plus grand pour les données images, un peu moins grand pour les données textes et web, moyen pour les fichiers binaires.

Nous remarquons aussi d'après les résultats obtenus avec la variation de la taille du mot, et cela quelque soit le type de fichier que le taux de compression est meilleur lorsque la taille du mot est 4 pour les fichiers binaires et 5 pour les fichiers textes, web et images.

Cette différence entre les résultats est expliquée par le principe de notre méthode qui est basée sur les automates d'états finis déterministe, lors de la compression on construit tout d'abord l'automate pour tout le fichier on spécifie le nombre de transitions de l'état initial à l'état final qui correspond à la taille du mot. Après la compression on obtient un fichier compressé qui contient l'entête (arbre) plus les codes des mots du fichier source. A chaque fois que la taille du mot augmente la taille de l'automate augmente et on aura moins de mots (codes), ce qui implique l'obtention d'un taux de compression significatif.

Lorsque la taille du fichier source est importante la taille de l'entête est négligeable et la taille du fichier compressé dépondra uniquement du nombre de mots dans le fichier source. Sachant que :

$$\boldsymbol{nombre\ de\ mots = \frac{La\ taille\ du\ fichier\ source}{la\ taille\ du\ mot}}$$

Donc, on a à chaque fois que la taille du mot augmente le nombre de mots diminue par conséquent la taille du fichier compressé. Ce qui donne des taux élevés.

IV.5 Conclusion :

Dans ce chapitre nous avons présenté au premier lieu l'environnement de développement de notre travail, dans la deuxième partie, nous avons donné les résultats obtenus après les tests effectués sur les différents type de fichier : texte, web, image, binaire avec la variation de la taille du mot, ainsi nous avons fait une comparaison entre le taux de compression obtenu pour ces différents type de fichier ainsi les interprétations des schémas, suivi par la justification de ces résultats.

Conclusion générale

Conclusion générale

Nous avant présenté dans ce mémoire quelques méthodes de compression de données à savoir Huffman, Shannon, LZW et LRE et méthodes par automate à états finis. Pour situer notre travail, nous avons exposé les outils nécessaires à notre approche de compression par une présentation de quelques généralités, définitions et notions de base sur notre domaine d'étude.

Notre travail a pour objectif d'implémenter une nouvelle méthode de compression de données avec les automates d'états finis afin de réduire le volume de données pour gagner d'espace mémoire.
On a constaté que cette méthode proposée donne des bons résultats pour tout type de données : images, textes, binaires et web après avoir effectuer des tests avec la variation de la taille du mot.

Ce travail nous a permis d'acquérir beaucoup de connaissances dans un domaine aussi vaste qui est la compression de données, des différentes méthodes de compression de données, ainsi les algorithmes utilisés et de mieux se familiariser avec le langage de programmation C++. Le travail que nous avons fait peut être amélioré pour donner des résultats plus meilleurs par exemple en faisant varie la taille du mot pour des valeurs plus grandes.

Bibliographie

Bibliographie

[1] : codage, cryptographie et application ; Bruno Martin ; Presse Polytechnique et universitaire romandes, 2004.

[2] : Téléinformatique I et III ; Henri Nussbaumer ; presse polytechnique romandes

[3] : www.syscope.net/info/info.htm

[4]: www-igm.univ-mlv.fr/ beal/ Enregistrement/TheorieInfo

[5]:http:// www.iict.ch/tcom/cours/PDF/compression.pdf

[6] : MM. S.Maadi, Y. Peneveyre, et C. Lambercy « Compression de données avec pertes ».

[7] : THEORIE DESCODES : compression, cryptage, correction(2005) par Yves denneulin, jean- guillaune durmas…

[8] : théorie de l'information codage source-canal ESCPI-CNAM version 24 janvier 2007.

[9] : A.nouri et H.Djoubani : « la compression de données », mémoire d'ingénieur, UMMTO 1999

[10] : la compression des données multimédias BENTHANI NADA FISSRIRI SOUFIANE 2007/2008

[11] : compression de données PEREIRA Vincent- Le PRETTE Frank- HACAULT Vincent décembre 2004

[12] : Alexandre THIL-Master1-Informatique-université de Metz Initiation à la recherche Algorithmes de compression de données et de Traitement d'images

[13] :R.Chabbaraka et M.Bekkouche « conception et réalisation d'un logiciel pour la compression et cryptage des informations multimédias », mémoire d'ingénieur, UMMTO 2002

[14] : Science appliquées-compression numérique & taux de transferts-les images réalisées par Sait-Moulin

[15] : facultés universitaires notre dame de la paix département Education et technologie images, sons, vidéos Monique colinet.

[16]: Mark nelson. Compression de données (images, son, texte).édition Dunode 1992.

[17] : Grégoire MERCIER '' Notion d'information et codage sans perte ''.

[18] : Compression de dictionnaires électroniques
Lamia Tounsi, Béatrice Bouchou, Christophe Lenté, Denis Maurel
Université François Rabelais Tours, Laboratoire d'Informatique, France

[19] : Tounsi L. (2007). Sous-automates à nombre fini d'états. Application à la compression De dictionnaires électriques. Thèse de doctorat, Université François Rabelais de Tours

[20] : Tounsi L., Lenté C. and Maurel D. (2006). Analyse statistique de la structure des Automates représentant des dictionnaires électroniques. In *Proc. of JADT'06*, pages 527-935.

[21]: Maurel D. and Guenthner F. (2006). Automata and Dictionnaries. King's College Publications.

[22]: Wikipedia