REPUBLIQUE ALGERIENNE DEMOCRATIQUE ET POPULAIRE
MINISTERE DE L'ENSEIGNEMENT SUPERIEUR ET DE LA RECHERCHE SCIENTIFIQU

UNIVERSITE MOULOUD MAMMERI DE TIZI-OUZOU
FACULTE DE GENIE ELECTRIQUE ET INFORMATIQUE

DEPARTEMENT D'INFORMATIQUE

Mémoire de master

En informatique

Option : Conduite d'un projet informatique

Méthode de compression de données sans perte basée sur les nombres premiers

Proposé et Dirigé par :

M^r^ SADOU.S

Réalisé par :

M^elle^ OUAISSA Nassima

Promotion 2011/2012

Remerciement

Remerciement

Je tiens à remercier vivement mon promoteur M^R SADOU.S pour m'avoir encadré et guidé tout au long de mon projet. Je le remercie surtout pour sa disponibilité.

Mes sincères salutations aux membres du jury qui me feront l'honneur de juger mon travail.

Dédicace

Dédicaces

Je dédie ce modeste travail :

- A mes très chèrs : mon père et ma mère aux quels je dois tout, mon frère et mes sœurs.

- A toute ma famille et tous mes amis (es).

Une dédicace spéciale à la mémoire de mon très cher « Dada ».

Table de matière

CHAPITRE III: Analyse et conception

CHAPITRE IV: Test et implémentation

Table de figures

Liste des tableaux

Introduction Générale

Introduction générale

Les développements technologiques et les exigences des utilisateurs entraînent le traitement de quantités de données; 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.

La compression de données consiste à réduire la taille de l'information pour son stockage et son transport, pour que le temps de transfère des fichiers soit plus court et donc à moindre coût.

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.

Pour mener à terme notre travail et à fin de rendre la démarche compréhensible, le présent mémoire est structuré de la manière suivante :

- Le premier chapitre, présentera des généralités de la compression de données.
- Le second chapitre décrira le nouvel algorithme de compression de données
- Le troisième chapitre est consacré pour l'étude et la conception de la méthode décrite dans le deuxième chapitre.
- Le dernier chapitre détaillera l'implémentation ainsi que le résultat des différents tests effectués sur cette méthode.

Chapitre I:

Généralités

1. Introduction :

La compression des 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. Ce chapitre a pour objectif de donner un bref aperçu de quelques généralités de la compression de données.

2. Définition [1] :

La compression de données, c'est l'ensemble des techniques et des méthodes que l'on utilisera pour réduire le volume de données sans perdre d'information importante.
Elle est très utile pour plusieurs applications informatiques. Réduire le volume des données permet d'emmagasiner plus d'information sur un même média, ou prendre moins de temps pour les transferts. Dans certain cas, le volume de données est tel qu'il serait quasiment impossible de le gérer sans utiliser la compression.

Les différents algorithmes de compression sont basés sur 3 critères :

– **Le taux de compression [14] :** c'est le rapport de la taille du fichier compressé sur la taille du fichier initial. Il sert à évaluer la qualité de la compression en faisant l'opération suivante:

$$T=1 - \frac{Taille\ compressée}{Taille\ source} \times 100$$

– **La qualité de compression :** sans ou avec pertes (avec le pourcentage de perte).
– **La vitesse** de compression et de décompression.

Un compresseur utilise un algorithme qui sert à optimiser les données en fonction du type de données à compresser ; un décompresseur est donc nécessaire pour reconstruire les données grâce à l'algorithme dual de celui utilisé pour la compression.
La méthode de compression dépend du type de données à compresser car une image ou un fichier audio ne représentent pas le même type de données.

3. Quelques éléments de la théorie de l'information

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

3.1 Quantité d'information

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

Soit *S*, une source d'information (alphabet) $S=(X_1, X_2, X_3,\ldots, X_k)$ de *K* symboles. Et soit $P(X_i)$ la probabilité d'apparition de X_i.
La quantité d'information (I_i) contenue dans un flot de données est une fonction de l'inverse de la probabilité d'apparition de ce symbole, elle est donnée par la relation suivante :
$I_i= \text{Log}_2 (1/P(X_i))= -\text{Log}_2 (P(X_i))$
L'unité de la quantité d'information est le Shannon

3.2 Entropie

En général, l'entropie mesure le degré de désordre dans un système donné.
En théorie d'information, l'entropie mesure le degré du hasard dans un flot de données (fichier) Un fichier qui a un faible taux d'entropie, dont les éléments sont tous prévisibles à l'avance, sera beaucoup compactable. Par exemple un fichier rempli de zéros. La quantité d'information qu'il contient est nulle, le hasard inexistant, l'entropie minimale.
Au contraire un fichier zippé (déjà compacté) contient des octets apparemment aléatoires sans aucune relation les uns avec les autres. La quantité d'information y est maximale, presque par définition puisqu'il s'agit d'un fichier déjà compressé, donc dont on a à priori supprimé toutes les redondances. Il contient par ailleurs une dose de hasard maximale, soit une entropie maximale.
L'entropie H d'un fichier est l'information moyenne contenue par chaque symbole, elle est donnée par la relation

$$H=\Sigma I_i P(Xi)=- \Sigma P (xi) \text{Log2} (P(Xi)$$

Propriétés

- $0 \leq H \leq \text{Log}_2 (n)$
- H=0 lorsque P(Xi)=1
- $H=\text{Log}_2(n)$ lorsque P(Xi)=1/n, i=1..n

3.3 La redondance

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

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

4. Techniques de codage

Le codage est une fonction qui fait correspondre à chaque symbole (ou groupe de symboles) du flot de données à compresser des symboles compacts (codes).

- ✓ **Caractéristique d'un code**

Pour avoir un bon codage des données, le code doit vérifier les propriétés suivantes :

- Tous les mots du code peuvent être distingués.
- Le décodage ne doit pas donner lieu à aucune ambiguïté.
- Aucun mot du code n'est un sous mot initial d'un autre.

Dans la section suivante; on verra un moyen de codage qui répond aux différentes conditions citées ci-dessus, ce codage fait appel à la théorie des nombres premiers.

5. Classification des algorithmes de compression:

5.1. Compression symétrique / asymétrique [13] :

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). Et pour lesquelles on ne veut pas perdre de temps à archiver.

Figure I.1: Compression de type symétrique.

5. 2. Compression physique / logique [3] :

On considère généralement la compression comme un algorithme capable de comprimer énormément de données dans un minimum de place (*compression physique*), mais on peut également adopter une autre approche et considérer qu'en premier lieu un algorithme de compression a pour but de recoder les données dans une représentation différente plus compacte contenant la même information (*compression logique*).

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

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

La compression logique est accomplie à travers le processus de substitution logique qui consiste à 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 chaîne "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 même des données.

5.3. La compression statistique / numérique [2] :

Pour les algorithmes qui travaillent au niveau statistique, la valeur des motifs ne compte pas. Ce sont les probabilités qui comptent, et le résultat est inchangé par substitution des motifs tandis que pour ceux qui opèrent au niveau numérique, les valeurs des motifs influent sur la compression (par exemple JPEG), et les substitutions sont interdites.
Enfin le critère de classification le plus pertinent est basé sur la perte des données.

5.4. La compression sans / avec perte [1]:

5.4.1. Compression sans perte (lossless)

La compression sans perte est particulièrement applicable aux données qui demandent une restitution exacte. Dans le contexte de la compression sans perte la méthode prends en entrée une série de bit X qu'elle transforme en une série de bits Y plus courte que X.la série de bits Y transmise ou stockée pour usage ultirieur.lorsque l'on veut récupérer les données, on prend Y et on applique la méthode de compression inverse -- la méthode de décompression--pour récupérer X intact. La figure suivante illustre schématiquement le processus.

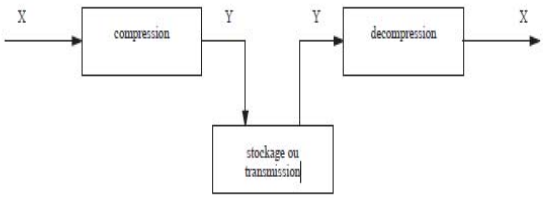


Figure I.2 : Une vision simplifiée de la compression sans perte

Les algorithmes de compression sans perte (connu aussi sous le nom de non destructible, réversible, ou conservative) sont des techniques permettant une reconstitution exacte de l'information après le cycle de compression / décompression. Aucune donnée n'a été perdue ou oubliée. Les données n'ont pas été modifiées.

A)

Il existe 3 types d'algorithmes de compression sans perte [2] :

A.1) Codage statistique :

Le but est de :

– Réduire le nombre de bits utilisés pour le codage des caractères fréquents.

– Augmenter ce nombre pour des caractères plus rares.

Exemple

Certaines informations sont plus souvent présentes que d'autres dans les données que l'on veut compresser.

Dans un fichier HTML par exemple, on trouvera beaucoup de signes <, /, et >. On va chercher à coder les données se répétant souvent sur moins de bits, et les données moins fréquentes sur plus de bits. On va chercher à élargir la réduction des répétitions des groupes d'octets plutôt que des octets simples.

A.2) Utilisation d'un dictionnaire :

Le but est de :

– Réduire le nombre de bits utilisés pour le codage des mots fréquents.

– Augmenter ce nombre pour des mots plus rares.

A.2.1. Lempel-Ziv 1977 (LZ77) [5]

La compression LZ77 remplace des motifs récurrents par des références à leur première apparition.

Elle donne de moins bons taux de compression que d'autres algorithmes (PPM, CM), mais a le double avantage d'être rapide et asymétrique (c'est-à-dire que l'algorithme de décompression est différent de celui de compression, ce qui peut être exploité pour avoir un algorithme de compression performant et un algorithme de décompression rapide).

LZ77 est notamment la base d'algorithmes répandus comme Deflate (ZIP, gzip) ou LZMA (7-Zip)

A.2.2. Lempel-Ziv 1978 et Lempel-Ziv-Welch (LZ78 et LZW) [5]

LZW est basée sur la même méthode, mais Welch a constaté que, en créant un dictionnaire initial de tous les symboles possibles, la compression était améliorée puisque le décompresseur peut recréer le dictionnaire initial et ne doit donc pas le transmettre ni envoyer les premiers symboles. Elle a été brevetée par UNISYS et ne pouvait donc être utilise librement dans tous les pays jusqu'à l'expiration du brevet en 2003. Elle sert dans les modems, mais UNISYS s'est engagé à vendre une licence à tout fabricant avant d'être acceptée comme norme de compression internationale pour les modems.

La compression Lempel-Ziv-Welch est dite de type dictionnaire. Elle est basée sur le fait que des motifs se retrouvent plus souvent que d'autres et qu'on peut donc les remplacer par un index dans un dictionnaire. Le dictionnaire est construit dynamiquement d'après les motifs rencontrés.

A.3) Substitution de séquences :

Comprime les séquences de caractères identiques.

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

B) **Quelques méthodes de compression sans perte**

Voici quelques méthodes statistiques de compression sans perte [4] :

B.1 .Run-length Encoding (RLE) [3]

RLE est un algorithme de compression de données qui est utilisé par la plupart des formats de fichiers bitmaps tels que TIFF, BMP et PCX. Il a été créé pour compresser n'importe quel type de données sans tenir compte de l'information qu'elle contient. Toutefois le contenu des données va affecter le taux de compression qu'il pourra atteindre. Bien que la plupart des algorithmes RLE ne puissent pas atteindre les forts taux de compression d'autres méthodes de compression plus avancées, RLE est à la fois simple à implémenter et rapide d'exécution ce qui fait de lui une bonne alternative entre utilisé un algorithme de compression plus complexe ou laisser l'image sans compression.

RLE est l'une des méthodes les plus anciennes, les plus simples et la plus utilisée. Tout son secret consiste à identifier et supprimer des redondances d'informations en les codant sous une forme plus compacte.

Exemple :

Non compressé, le passage comprenant 15 caractères "A" devrait normalement prendre 15 bytes à stocker.
AAAAAAAAAAAAAAA
La même chaîne après codage ne prend plus que 2 bytes : **15A**

Le "15A" généré pour représenter la chaîne de caractère est appelé un paquet RLE (*RLE packet*). Ici, le compteur de passage contient le nombre de répétitions soit 15. Le deuxième byte "A", la valeur du passage, contient la valeur répétée dans le passage.

Un nouveau paquet est généré à chaque fois que le caractère change ou chaque fois que le nombre de caractère dans le passage excède la valeur maximum que peut prendre le compteur. Si l'on suppose que notre chaîne de 15 caractères contient maintenant 4 passages différents : AAAAAAbbbXXXXXt

En utilisant le codage RLE, cette chaîne pourra être compressée en 4 paquets de 2 Bytes : 6A3b5X1t

Ainsi, après le codage, la chaîne de 15 bytes de départ prendra seulement 8 bytes de données pour représenter la chaîne. Dans ce cas, RLE permet d'atteindre un taux de compression de 2:1.

Mais regardons maintenant ce qui se passe sur l'exemple suivant constitué principalement de caractères uniques : Cannibalisation
Après le codage RLE :
1C1a2n1i1b1a1l1i1s1a1t1i1o1n

On peut déduire de tout cela que le codage RLE est simple et efficace mais que l'efficacité de la compression dépend fortement du type de données à encoder. Une image en noir et blanc constituée d'une grande partie de blanc s'encodera très facilement en raison de la grande quantité contiguë de données identiques. A l'inverse, une image photographique constituée de beaucoup de couleurs différentes s'encodera très difficilement.

B.2. Le codage de Shannon

Le principe du codage statistique remonte à la fin des années quarante et repose sur les travaux du mathématicien Claude Shannon (laboratoires Bell), qui démontrait l'existence d'une méthode permettant de compacter les flux d'information sans rien perdre de leur signification.
L'idée directrice était donc que tout message est porteur d'une certaine densité d'information ou entropie dont le calcul peut s'effectuer en proportion de leur fréquence probable d'apparition. Cette entropie ne correspond pas forcément à la taille du message, dans la mesure où la plupart des messages contiennent de nombreuses redondances, ce qui permet d'en prévoir les séquences répétitives. Si l'on parvenait à calculer la fréquence probable d'apparitions des informations d'un message, il deviendrait du coup tout aussi simple d'en éliminer les redondances et de les encoder de manière efficace.

Plus un symbole est courant, plus il est aisé d'en prévoir l'apparition. Moins il nécessite d'informations pour l'identifier, plus son entropie est élevée. Donc s'il était codé en fonction de cette dernière, il pourrait occuper moins d'espace qu'un caractère plus rare dont la prévision est plus délicate.

e	11	n	6,6	m	3	f	1,22	j	0,17	ç	0,04
s	9,27	o	6,04	p	2,79	v	1,01	z	0,12	w	0,03
i	8,61	é	4,87	d	2,23	q	0,88	k	0,12	ô	0,03
a	7,8	l	4,75	g	1,91	y	0,65	ï	0,08	î	0,03
r	7,42	u	4,49	h	1,67	x	0,41	â	0,08	û	0,03
t	6,8	c	3,87	b	1,64	è	0,3	ê	0,06	à	0,01

Figure I.3 : Fréquence d'apparition des lettres en français

Cette répartition a été faite à partir d'un dictionnaire mais il est clair qu'elle peut varier en fonction du texte analysé (texte littéraire, texte commercial…).

Exemple :

On peut néanmoins se baser, **pour un message donné**, sur la théorie de l'information pour déterminer les probabilités et en déduire un codage approprié.

Nous utiliserons comme exemple, le message "BANANES ET ANANAS". Les propriétés statistiques de ce message sont données à la (figure I.4). Selon cette figure, on peut constater qu'un codage optimal selon Shannon doit permettre de coder le message en exactement 43.945 bit, contre 136 en ASCII. D'autre part, on doit être en mesure de coder le symbole A avec moins de 2 bit (1.765 bit très exactement), à condition de coder le symbole T avec 4.08 bit.

s	f(s)	p(s)	H(s)	H	ASCII
A	5	0.294	1.765	8.825	40
N	4	0.235	2.09	8.36	32
E	2	0.118	3.1	6.2	16
S	2	0.118	3.1	6.2	16
<space>	2	0.118	3.1	6.2	16
B	1	0.059	4.08	4.08	8
T	1	0.059	4.08	4.08	8
Total	**17**		**21.315**	**43.945**	**136**

Figure I.4: Caractéristiques statique de **"BANANES ET ANANAS"**

s : symbole

f(s) : fréquences d'apparition du symbole

p(s) : probabilité d'apparition du symbole

H(s) : quantité d'information convoyée par le symbole

H : quantité d'information cumulée par ce symbole (f(s) * H(s))

ASCII : quantité de décision selon un code ASCII

Plusieurs méthodes ont été développées pour utiliser les propriétés statistiques des messages. Nous allons étudier les deux plus classiques :

B.3.Le codage Shannon-Fano [1]

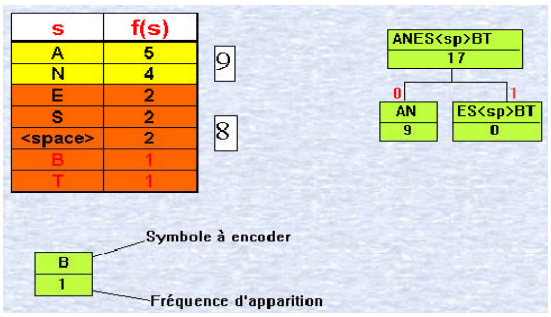
Shannon et Fano ont introduit séparément le même algorithme pour calculer un code efficace à partir d'une distribution arbitraire. Voici comment cet algorithme se procède Les symboles sont d'abord tries en ordre décroissant de fréquence .ce nouvel arrangement des codes formera la liste que nous utiliserons pour le reste de l'algorithme .au début tous les codes sont initialises comme étant la séquence vide. Ensuite cette liste est devises en deux parties (sans changer l'ordre des symboles)de façon à ce que la somme des fréquences dans la première partie soit la plus égale possible à la somme des fréquence de la seconde partie. A tous les codes des symboles de la première partie de la liste on ajoute 0 à la fin, et 1 à tous les codes des symboles de la seconde partie. On répète la même procédure, récursivement, dur les deux sous listes, jusqu'à ce que nous atteignions des listes n'ayant qu'un élément.

Cette procédure nous fournit un code uniquement décodable qui la caractéristique d'être structuré en arbre (tree structured code).

Il y'a des problèmes avec cet algorithme .un de ces problème est qu'il n'est pas toujours possible d'être certain ou couper la liste en deux .couper de façon a minimiser la différence.

Un exemple de codage selon Shannon-Fano [4]

Nous allons effectuer la compression de notre message-témoin en utilisant l'algorithme de Shannon-Fano. La première subdivision est indiquée à la figure2.10, page 41. On remarque que, dans l'arbre de codage résultant, on inscrit un "0" en regard de la branche de gauche, et un "1" en regard de la branche de droite. Ce codage est sans effet sur l'efficacité du résultat, et est donc arbitraire.



+

Figure I.5: Première subdivision selon

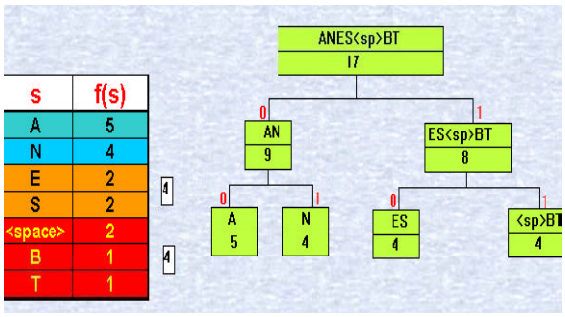


Figure I.6 : Deuxième subdivision

Shannon-Fano

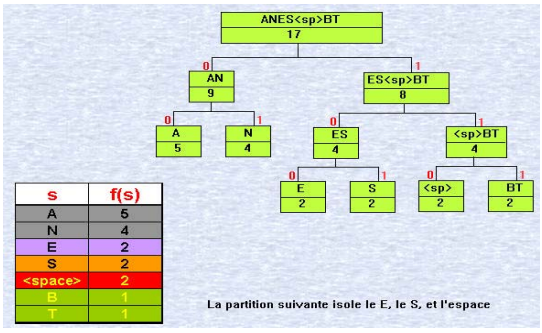


Figure I.7: Troisième partition

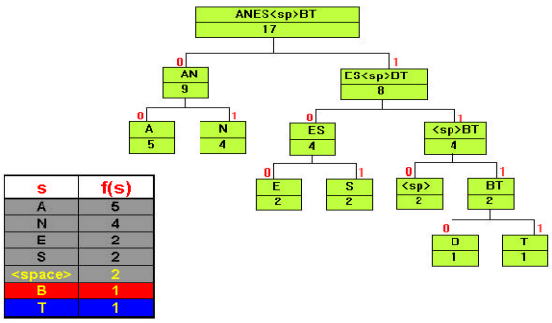


Figure I.8: Partition finale

Pour connaître 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 logique nécessaire.

B.4. Le codage de Huffman

L'étude de David Huffman (1952) se fonde sur l'idée que certains caractères sont susceptibles d'apparaître plus souvent que d'autres dans un fichier, laissant la possibilité de les coder sur un nombre de bits plus restreint. Mais il inverse le raisonnement de Shannon et Fano. Ainsi plutôt que de subdiviser une liste indéfiniment avec pour effet de développer un arbre à partir de son sommet, pourquoi ne pas le construire en partant d'en bas ?

Il eut l'idée d'examiner la liste des caractères pour trouver les deux moins fréquents, de

créer un nouveau symbole parent des deux premiers et somme de leurs comptes de fréquences, et de l'ajouter à la liste. Les symboles enfants prennent alors respectivement les valeurs 0 et 1 et sont retirés de la liste. Le processus est répété sur l'ensemble des caractères jusqu'à ce qu'il ne reste plus qu'un seul parent formant la racine de l'arbre. Pour décoder un symbole, il suffit de descendre dans cette arborescence. Cette méthode garantit en général un gain de 40 % d'autant plus intéressant que le processus s'effectue rapidement, ne nécessite que très peu de mémoire et ne peut en aucun cas créer de situation désespérée, ni de perte d'espace, malgré le petit en-tête reporté dans la table de sortie.

Exemple de codage selon Huffmann

Reprenons l'exemple précédent, à savoir "BANANES ET ANANAS". On peut voir sur la table de fréquences suivante (figure I.9) des lettres de ce message, bien sûr identique à celle que nous avions obtenue pour le codage selon Shannon-Fano. On va commencer le partitionnage par les lettres les moins fréquentes, à savoir dans ce cas, B et T, pour former un symbole combiné BT, qui aura en toute logique une fréquence d'apparition de f(B) + f(T), c'est-à-dire 2. Cette fréquence pourra à son tour se comparer à la fréquence d'apparition de E, S et <space>, et ainsi de suite jusqu'à ce que l'on obtienne un symbole combiné contenant le total du message, soit f(s) = 17. A mesure de la construction de l'arbre, on notera avec un "0" la branche de gauche, et un "1" la branche de droite (choix arbitraire).

BANANES ET ANANAS

s	f(s)
A	5
N	4
E	2
S	2
<space>	2
B	1
T	1
Total	17

Figure I.9 : Table de fréquences des lettres de message

La première partition est présentée à la figure suivante :

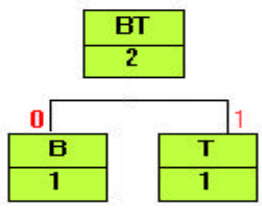


Figure I.10 : Première partition

On groupe ensuite les symboles ayant un poids (fréquence d'apparition) de 2 ensembles, pour aboutir à l'arbre partiel de la figure I.11.

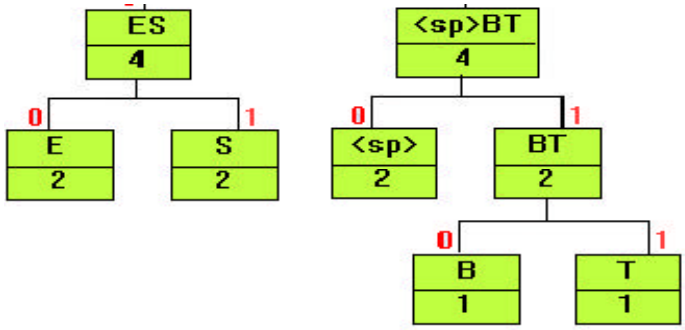


Figure I.11: Deuxième partition

Les figures I.12, et figure I.13 donnent les deux dernières étapes du codage selon Huffmann. On constate facilement que l'on parvient à un codage équivalent à celui de Shannon-Fano.

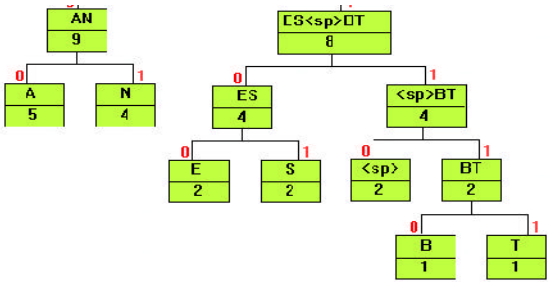


Figure I.12: Troisième partition

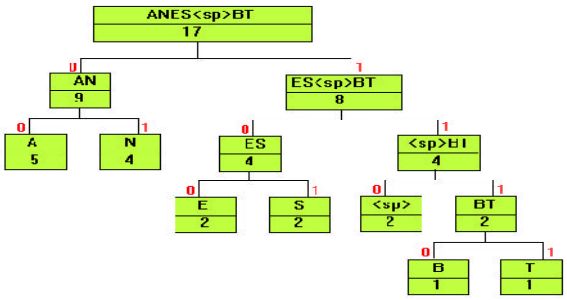


Figure I.13: Partition finale

5.4.2. Compression avec perte (lossy)

Avec la compression avec perte, on acceptera que les données restituées soient une approximation satisfaisante des données originales. La figure suivante illustre le processus général.ici on compresse une serie de bits X pour obtenir la série Y, qui est transmise ou stickée.lorsque on décompresse Y on retrouve Z, qui est potentiellement différent de X .Si Z diffère de X il faudra alors que Z ~ X, selon une mesure appropriée.

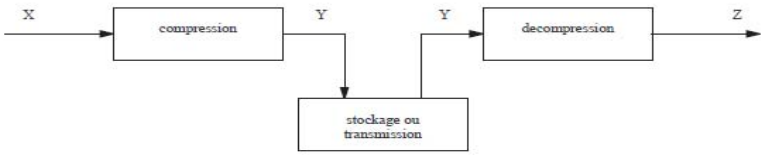


Figure I.14: Une vision simplifiée de la compression avec perte

Son principe est basé sur l'étude précise de l'œil et de l'oreille humaine. Les signaux audio et vidéo contiennent une part importante de données que l'œil et l'oreille ne peuvent pas percevoir et une part importante de données redondantes.
Les objectifs de la compression avec pertes sont d'éliminer les données non pertinentes pour ne transmettre que ce qui est perceptible et, comme pour la compression sans perte, d'éliminer l'information redondante.

Quelques méthodes de compression sans perte

Il existe des algorithmes de compression consacrés à des usages particuliers, dont en voici 3 :

– Compression du son (Audio MPEG, ADCPM ...).
– Compression des images fixes (JPEG,...).
– Compression des images animées (MPEG, ...).

6. Performances et efficacité [2] :

Nous allons nous intéresser à une série d'exemples afin de montrer le gain de compression dans des domaines informatiques.

Gain en temps sur les lectures / écritures :

– Temps pour écrire une image de 14Mo : 5s (soit 2,8Mo/s).
– Temps pour compresser une image de 14Mo en JPEG : 3,4s, soit 4,11Mo/s, résultat : 503Ko (soit 4% de l'original)

– Temps pour écrire une image JPEG de 503Ko : 0,2s
– Temps total compression + écriture : 3,6s, soit 1,4s de moins (gain de 28% en temps et 96% en volume stocké sur disque)

Gain sur les temps de transmission :

– Temps de transmission. 56Kbps/s d'un fichier HTML de 30Ko : 5,4s
– Temps de compression du fichier HTML au format« GZIP » : 0,05s (soit 600Ko/s), taille compressée : 15Ko (gain de 50% en volume)
– Temps de transmission de l' HTML compressé : 2,7s
– Temps total compression /transmission/ décompression : 2,8s, au lieu de 5,4s, soit un gain de près de 50% en temps.

Ainsi, nous avons pu voir que les méthodes de compression sont très utiles et apportent parfois des résultats spectaculaires. Sachant le fichier à compresser, chaque algorithme a ses avantages et ses inconvénients et il faut donc choisir l'algorithme en fonction du type de données.

7. Conclusion :

Ce chapitre est dédié à l'initialisation et la familiarisation avec la compression de données. Différents algorithmes de compression de donnée ont été définis, afin d'exploiter au maximum les ressources de stockage et de transmission. Les systèmes actuels procèdent ainsi à l'encodage des messages faisant appel au plus petit nombre de bits possibles, pour une meilleure optimisation.

Chapitre II:

Présentation de la méthode de compression

1. Introduction :

Plusieurs méthodes de compression ont vu le jour depuis que la discipline est apparue ; cette variété de méthodes est due à la différence des types de données ciblés par cette compression : images, son, texte,…etc.

Dans ce chapitre nous allons proposer un algorithme de compression qui repose sur un modèle mathématique (théorie des nombres premiers).Cet algorithme peut être utilisé aussi bien pour résoudre le problème de l'espace de stockage que pour résoudre celui du débit limité surtout lorsqu'il s'agit des applications temps réel.

2. Quelques notions sur les nombres premiers: [7]

Dans cette section on se limetra à l'ensemble des entiers N

a. Définition:

Un entier naturel p est dit PREMIER si ces seuls diviseurs dans N sont lui même et 1. On considère que 1 n'est pas un nombre premier (convention), la liste des nombres premiers est alors est: 2,3,5,7,11,13,17,19,23,….etc.

La définition d'un nombre premier en N peut s'annoncer de la façon suivante:

p est premier si et seulement si pour tous couple de (a,b) de N, on a:

$$p \text{ est devisible par } ab \Leftrightarrow \begin{cases} P = a \text{ ou } P=b \quad \& \\ A =1 \text{ ou } b=1 \end{cases}$$

b. Proposition:

1. Tout entier naturel different de 1 admet au moin un diviseur premier

2. Tout entier naturel p different de 1 non premier admet un diviseur premier a tel que $a \leq \sqrt{p}$

3. IL existe une infinité de nombre premier.

4. Soit $n \geq 2$ un entier. si n ni devisible par aucun nombre premier p tel que $p^2 \leq n$, alors n est premier .

c. Thèoreme fondamentale de l'arithmétique:[6]

Tout entier naturel non nul est de façon unique (à l'odre près de facteurs) produit d'un nombre fini de nombres premiers, c'est à dire tout nombre entier naturel est décomposé de façon unique en produit de ces diviseurs premiers

- Les facteurs premiers sont des nombres premiers
- Mise à part 2 , tous les nombres premiers sont impairs
- Tout nombre premier impair est de la forme 4k+1 ou 4k+3

d.Thèorem d'Euclide [7]

L'ensemble des nombres premiers est infini.

Trois proprieties fondamentales:

1. Il n'existe pas de formule algébrique pour représenter un nombre premier.

2. Il existe une infinite de nombres premiers.

3. La factorisation d'un nombre en facteurs premiers est unique.

e. Décomposition en produit de nombres premiers:

Pour tout entier naturel *n*, il existe une suite unique (a_1, a_2, a_3, …, a_k…) d'entiers contenant un nombre fini de terme non null telle que:

$$n = p_1^{a1} \times p_2^{a2} \times p_3^{a3} \times \ldots . \times p_k^{ak} \times \ldots . = \prod_{i=1}^{+\infty} p_i^{ai} = \prod_{p, \in p} p_i^{ai}$$

Ou $\mathbf{P_i}$ est les facteurs premiers et $\mathbf{a_i}$ leurs puissances.

C'est la décomposition en facteur premier de *n.*

3. Principe du nouvel algorithme

Le principe consiste à l'utilisation des propriétés mathématique des nombres naturels qui admettent une décomposition unique en nombres premiers.

On a $\forall n \in N, \exists P_1, P_2, \dots P_i, Tel\ que\ P_i\ est\ premier\ et\ n = P_1 \times P_2 \times, \dots \times P_i$.

3.1. Compression : cette propriété est exploitée dans le processus de compression selon la démarche suivante :

- Affecter pour chaque symboleS_i, du fichier à compresser, un nombre premier P_i .
- Composer ces nombres premiers pour avoir des nombres naturels qui représentent le résultat de compression

Exemple :

Soit le message « data …. » qu'on souhaite compresser.

1- Affectation des nombres premiers aux différents Symboles, pour cette étape il est intéressant d'affecter les nombres premiers les plus petits pour les symboles les plus fréquents.

Symboles	Fréquences	Nombre premier
a	2	2
d	1	3
t	1	5
espace	1	7
.	.	.
.	.	.

Tableau II.1: Affectation des nombres premiers aux différents symboles

2- Résultat de compression :

Le message « data … » sera remplacer par :

$$N_1 = \underbrace{3^1}_{D} \times \underbrace{2^2}_{a} \times \underbrace{5^3}_{t} = 1500$$

$$N_2 = \underbrace{2^1}_{a} \times \underbrace{7^2}_{espace} = 98$$

Donc le résultat de la compression du message est N_1N_2

3.2. Décompression : le processus de décompression consiste à lire le contenu du fichier compressé, qui est constitué d'une série de nombres naturels N_i. Pour chaque nombre on effectue la décomposition en nombres premiers qui correspondent aux différents symboles de fichier d'origine.

Exemple : dans l'exemple précédent, on a obtenu un fichier compressé constitué des nombres naturels N_1N_2 avec N_1=1500, N_2=98.

-la décomposition en nombres premiers de N_1 et N_2 donne :

$$N_1 = \underbrace{3^1}_{D} \times \underbrace{2^2}_{a} \times \underbrace{5^3}_{t} \quad , \quad N_2 = \underbrace{2^1}_{a} \times \underbrace{7^2}_{espace}$$

Par conséquent le message « data » est restitué.

4. Amélioration de l'approche :

L'inconvénient de cette méthode est que les positions des symboles, du message à compresser, sont préservées grâce aux différentes puissances sur les nombres premiers utilisés dans le processus de compression ce qui rend les nombres naturels, résultant de la composition, très grand réduisant ainsi le nombre de symboles qui peuvent participer au processus de composition ce qui limite le taux de compression.

La solution est de chercher un moyen d'éliminer les exposants dans la formule de compression. L'idée est de trouver une fonction qui va permettre de coder l'ordre des symboles dans le message qui remplacera ainsi la technique des exposants.

Cela signifie que le résultat de la compression d'un message donné sera constitué d'une entité possédant deux attributs, un nombre naturel correspondant au produit des nombres premiers le constituant et un autre qu'on nomme Ordre qui permet de reconstitué l'ordre des symboles dans le message.

4.1. Illustration :

Soient les deux messages « ABC » et « BAC » à compresser, et supposons que les nombres premiers 2, 3, et 5 sont respectivement attribués à A, B et C.

4.2. Compression :

Le résultat de compression du message « ABC » est un enregistrement de deux attributs N_1, Ord_1 tel que N_1 représente le produit des nombres premiers correspondant ($N_1 = \underbrace{2}_{A} \times \underbrace{3}_{B} \times \underbrace{5}_{C} = 30$) et on remarque que le produit seul n'est pas suffisant pour garantir l'unicité du message lors de la décompression et cela est du à la commutativité de

l'opération de multiplication. Pour cela Ord_1 va permettre de conservé l'ordre des symboles. Donc le résultat final de la compression du message « ABC » est N_1 |Ord_1
Pareil pour le message « BAC », $N_2 = \underbrace{3}_{B} \times \underbrace{2}_{A} \times \underbrace{5}_{C} = 30$ et un autre attribut Ord_2 qui permet de reconstitué l'ordre initial des symboles dans le message « BAC ».

Remarque : les attributs N_i permettent de restituer les différents symboles du message sans plus autant indiquer leurs ordre d'où l'intérêt de l'attribut Ord_i.

Le tableau suivant illustre les différents cas possibles pour un message contenant les symboles A, B et C :

Message	Résultat de compression	
	N_i	Ord_i
ABC	2x3x5=30	Ord_0
ACB	2x5x3=30	Ord_1
BAC	3x2x5=30	Ord_2
BCA	3x5x2=30	Ord_3
CAB	5x2x3=30	Ord_4
CBA	5x3x2=30	Ord_5

Tableau II.2: Les différentes permutations pour un message de trois symbôles

L'objectif maintenant est de trouver cette fonction qui permet de coder l'ordre d'un vecteur donné.

4.3. La fonction de codage d'ordre :

Avant de présenter la fonction d'ordre, il est important de donner quelques caractéristiques de la relation Ordre d'un vecteur de N éléments.

- **Propriétés 1**: Pour N symboles distincts, ils existent N ! (permutations) ordres distincts

  Exemple 1: soit l'ensemble des symboles E= {A, B} alors on a N=2 éléments par conséquent le nombre d'ordre qu'on peut avoir est 2 ! = 2 ordres

Permutations	Ordre
A B	Ord_1
B A	Ord_2

Exemple 2: soit l'ensemble des symboles E = {A, B, C} alors on a N=3 éléments par conséquent le nombre d'ordre qu'on peut avoir est 3 ! = 6 ordres

Permutations	Ordre
A B C	Ord_1
A C B	Ord_2
B A C	Ord_3
B C A	Ord_4
C A B	Ord_5
C B A	Ord_6

- **Propriétés 2**: Chaque permutation possède son propre ordre (l'ordre est unique)

La codification de l'ordre d'une permutation donnée passe par les étapes suivantes :

Permutation➔ Code intermédiaire ➔ Code Ordre
Tel que le code intermédiaire est étroitement lié à la relation d'ordre.
Exemple :

Permutation	Code intermédiaire			Ordre
	C_3	C_2	C_1	
A B C	0	0	0	Ord_1=0 x 2!+0 x 1!+0x 0!=0
A C B	0	1	0	Ord_2=0 x 2!+1 x 1! +0x 0!=1
B A C	1	0	0	Ord_3=1 x 2!+0 x 1! +0x 0!=2
B C A	1	1	0	Ord_4=1 x 2!+1 x1! +0x 0!=3
C A B	2	0	0	Ord_5=2 x 2!+0 x1! +0x 0!=4
C B A	2	1	0	Ord_6=2 x 2!+1 x1! +0x 0!=5

Tableau II.3: Codification de l'ordre

Chaque digit C_i du code intermédiaire correspond à un symbole S_i de la permutation de façon à indiquer le nombre de symboles à droite de ce dernier qui sont inférieur à S_i.

Dans l'exemple ci-dessus, si on prend la permutation « B C A » alors le calcul de C_3, C_2 et C_1 se fait comme suit :

C_3 =1 : car il existe 1 seul symbole « A » qui est inférieur à B.

C_2=1 : car il existe 1 seul symbole « A » qui est inférieur à C.

C_1=0 : car il n'existe pas de symboles après le « A ».

Remarque : le digit C_1 est toujours à Zéro car il correspond au dernier symbole du vecteur et il n'existe aucuns symboles à sa droite. Donc on peut garder seulement 2 digits pour le code intermédiaire pour un vecteur de 3 symboles.

4.3.1. Généralisation :

Soit V = {S_1, S_2, ……. S_T} un vecteur de symboles de taille T, alors on a (T !) Permutations possibles qui donnent lieu à (T !) Codes d'ordre.

Pour chaque permutation (P_i), il existe un code intermédiaire $C = C_{T-1}C_{T-2} \ldots\ldots C_1$

Tel que C_i représente le nombre de symboles qui sont inférieur au symbole S_i et qui se trouvent à droite de ce dernier.

On note la fonction d'ordre pour une permutation P_i donnée : Ordre(P_i).

Alors on a :

$$Ordre(P_i) = C_{T-1} \times (T-1)! + C_{T-2} \times (T-2)! \ldots\ldots C_1 \times 1!$$

4.3.2. Propriétés de la fonction du codage d'ordre

- Unicité : chaque permutation possède un seul et unique code d'ordre.
- Le code ordre peut être considéré comme l'entropie d'ordre qui permet de mesurer le degré d'ordre d'un vecteur donné.
  Pour un vecteur totalement ordonné H_{min}=0 et la valeur de l'entropie est maximal H_{max} = (N ! -1) pour un vecteur totalement désordonné.
- La valeur de l'ordre est un entier varie de 0 à (N ! – 1) selon le degré d'ordre du vecteur.

4.4. Fonction de décodage d'ordre

Le résultat de la compression est une suite d'entités, chaqu'une est composée de deux attributs un entier naturels **N** qui représente le produit des nombres premiers correspondant aux différents symboles du message, et un autre attribut **Ord** qui représente le code d'ordre des symboles.

Pour retrouver les symboles du message, il suffit de faire la décomposition de N en nombres premiers et pour retrouver l'ordre des symboles dans le message on doit passer par le processus inverse de la fonction de codage d'ordre, c'est-à-dire, à partir du code Ord on retrouve le code intermédiaire par des divisions successives sur (N-1 !), (N-2 !), ….(1 !)

Exemple : soit E= {A, B, C, D}, dans ce cas le code d'ordre vari entre 0 et 4 !-1=23.

Prenant la valeur de Ord = 15 et essayant de retrouver l'ordre des symboles.

D'abord on cherche la valeur du code intermédiaire C= C_3 C_2 C_1

On a Ord = C_3 x 3! + C_2 x 2! + C_1 x 1! =15.

$C_3 = 15 / 3! = 2$, $C_2 = (15 - 2 \times 3!) / 2! = 1$ et $C_1 = (15 - 2 \times 3! + 1 \times 2!) / 1! = 1$ ➔ C= 2 1 1

Il reste le passage du code intermédiaire à l'ordre initial des symboles qui se fera selon l'algorithme suivant :

- Initialiser le message à une chaine vide, M= « »

- Trier les symboles par ordre croissant dans un vecteur

A	B	C	D
0	1	2	3

- C_3=2 cela signifie que le premier symbole est celui de la case 2 du vecteur qui est « C »

-Retirer le symbole « C » du vecteur et le concaténer avec le message M= « C »

A	B	D
0	1	2

- C_2=1 cela signifie que le symbole est celui de la case 1 du vecteur et qui est « B »

- Retirer le symbole « B » du vecteur et le concaténer avec le message M= « CB »

A	D
0	1

- C_1=1 cela signifie que le symbole est celui de la case 1 du vecteur et qui est « D »

- Retirer le symbole « D » du vecteur et le concaténer avec le message M= « CBD »

-il ne reste qu'un seul symbole qui est « A » qu'on concatène avec M ce qui donne le message final M= « C B D A »

Vérification :

M= « CBDA », C=C_3 C_2 C_1

C_3=2 car ils existent deux symboles «A » et « B » qui sont inférieur à « C »

C_2=1 car il existe un seul symbole « A » inférieur à « B »

C_1=1 car il existe un seul symbole « A » inférieur à « D »

Donc C= 2 1 1 par conséquent ordre(M)=2x3!+1x2!+1x1!= 15

5. Application

La méthode de compression proposée au début de ce chapitre, peut être améliorée en utilisant les deux propriétés, la décomposition en nombres premiers et le codage d'ordre.

5.1. Compression

La compression d'un fichier source consiste à le lire symbole par symbole et affecter pour chaque symbole un nombre premier selon sa fréquence, et il est intéressant d'affecter les plus petit nombre premier pour le symbole le plus fréquent.

Symboles	Fréquences	Nombre premier
S_1	F_1	2
S_2	F_2	3
S_3	F_3	5
.	.	.
.	.	.
.	.	.
S_N	F_n	P_n

Tableau II.4 : Affectation de nombres premiers aux symboles selon leurs fréquences

Remarque : Cette table doit être insérer dans le fichier résultant pour servir le processus de décompression. Après cela, il reste à codifier les symboles du fichier par les deux étapes suivantes :

- Etape 1 : Calculer le produit de nombres premiers correspondant aux symboles
- Etape 2 : Calculer le code d'ordre de la série des symboles utilisés dans l'étape 1

Exemple :

Soit M, un extrait d'un fichier avec M= « data compression …. », à compresser.

1. Etablir la table de correspondances Symbole : nombre premier

Symboles	Fréquences	Nombre premier
a	*2*	*2*
o	*2*	*3*
s	*2*	*5*
d	*1*	*7*
c	*1*	*11*
e	*1*	*13*
i	*1*	*17*
n	*1*	*19*
m	*1*	*23*
p	*1*	*29*
t	*1*	*31*
espace	*1*	*37*

2. Le résultat est une suite de couples$\{N_i|Ord_i\}$, N_i sur 16 bits et Ord_i sur 16 bits

   $N_1 = \underbrace{7}_{d} \times \underbrace{2}_{a} \times \underbrace{31}_{t} \times \underbrace{2}_{a} \times \underbrace{37}_{espace} \times \underbrace{11}_{c} > 2^{16}$, on ne peut pas le sauvegarder sur 16 bits donc on retire le dernier symbole qui est « c » du produit.

   $N_1 = \underbrace{7}_{d} \times \underbrace{2}_{a} \times \underbrace{31}_{t} \times \underbrace{2}_{a} \times \underbrace{37}_{espace} = 32116 < 2^{16}$, N_i est accepté.

3. Calculer le code de l'ordre du vecteur (7 | 2 | 31 | 2 | 37)

   Pour cela on calcule d'abord le code intermédiaire $C=C_4\ C_3\ C_2\ C_1$

   C_4=2, car ils existent deux nombres (2,2) à droite qui sont inférieur à 11

   C_3=0, aucun

   C_2=1, il existe un seul nombre (2) inférieur à 31

   C_1=0, aucun

   On a C= 2 0 1 0 alors $Ord_i = 2 \times 4! + 0 \times 3! + 1 \times 2! + 0 \times 1! = 50$

   Le résultat final de la compression du message M_1= « data » est (32116|50)

   Le reste du message M_2= « compression » est compresser de la même manière.

$$N_2 = \underbrace{11}_{c} \times \underbrace{3}_{o} \times \underbrace{23}_{m} \times \underbrace{29}_{p} = 22011$$

C= 1 0 0 ➔ $Ord_2 = 1 \times 3! + 0 \times 2! + 0 \times 1! = 6$

Le message « comp » est coder (22011|6)

5.2. Décompression

Le processus de décompression consiste à lire d'abord la table des fréquences des symboles à partir du fichier compressé puis lire les couples de valeur (N_i | Ord_i), et pour chaque couple décomposer le nombre N_i en nombres premiers et enfin décoder le code d'ordre Ord_i.

Exemple : On reprend l'exemple précédent, le fichier compressé contenait les couples de valeurs suivants : (32116|50) , (22011|6) ….etc

Le couple (32116 | 50) est composé d'un nombre N_1 =32116 et un code ordre Ord_1=50

1. Décomposer N_1 en nombres premiers : $N_1 = \underbrace{2}_{a} \times \underbrace{2}_{a} \times \underbrace{7}_{d} \times \underbrace{31}_{t} \times \underbrace{37}_{espace}$
2. Reconstitué l'ordre des symboles Ord_1=50

A	a	d	t	
0	1	2	3	4

- le code intermédiaire C= C_4 C_3 C_2 C_1

C_4 = (50 / 4 !) = (50/ 24) = 2

C_3= (50 - 2 x 4!) / 3! = (50-48)/ 3!= 0

C_2= (50- (2 x 4! + 0 x 3!))/2 != (2)/ 2!= 1

C_1= (50- (2 x 4! + 0 x 3!+1 x 2!))/1!=0/1 != 0

A partir de C = 2 0 1 0 on reconstitue l'ordre :

C_4=2 ➔ le symbole se trouve à la position 2 du vecteur trié. Le symbole est « d » , on le retire du vecteur :

A	a	t	
0	1	2	3

La valeur actuelle du message M est « d »

C_3=0 ➔ le symbole se trouve à la position 0 du vecteur trié. Le symbole est « a », on le retire du vecteur.

A	t	
0	1	2

La valeur actuelle du message M est « da »

C_2=1 ➔ le symbole se trouve à la position 1 du vecteur trié. Le symbole est « t » , on le retire du vecteur.

A	
0	1

La valeur actuelle du message M est « dat »

C_1=0 ➔ le symbole se trouve à la position 0 du vecteur trié. Le symbole est « a » , on le retire du vecteur.

0	1

La valeur actuelle du message M est « data »

Et le dernier symbole implicite est « » (espace) on l'ajoute à M

Le résultat final est M= «data » qui correspond parfaitement au message d'origine.

6. Avantage de la méthode [6] :

Cette méthode présente plusieurs avantages qu'on peut résume dans ces quelques points :

- **Simple à maître en œuvre (ne nécessite pas de dictionnaire) :** c'est une méthode qui n'utilise pas de dictionnaire, et cela réduit considérablement la taille du programme, ajouter a cela la simplicité du modèle mathématique utilisé.
- **Adéquate pour la compression des données temps réel :** le principe utilisé pour la Compression offre la possibilité de compresser un flot de données temps réel qui arrive caractère par caractère en lui appliquant la fonction de composition *F ()* au fur et à mesure que les octets arrivent sans attendre la constitution de la totalité du fichier à compresser . Dans cette configuration on voie bien que la charge de données transmise sur les lignes de communication sera aussi diminuée se permet d'avoir un débit virtuel plus grand (envoyer des

chaînes de caractères sous formes d'entiers de petite taille) et par conséquent des délais réduit qui vont donner une meilleur interactivité pour les applications temps réel.

Les inconvénients de cette méthode résident dans ses contraintes qui peuvent faire l'objet de travaux futurs pour en améliorer le rendement.

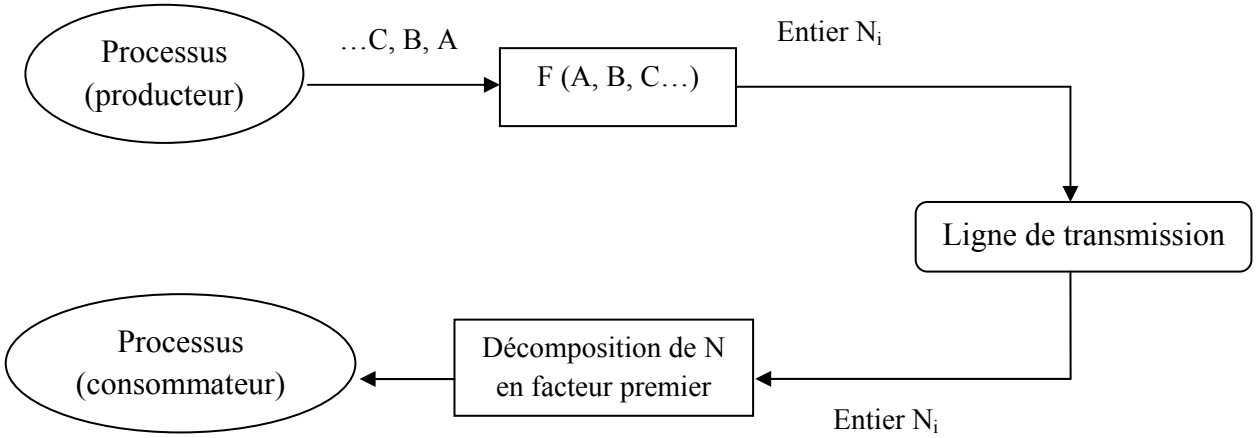


Figure II.1 : Compression temps réel

7. Conclusion :

Dans ce chapitre, nous avons proposé une nouvelle méthode pour la compression de données. Cette méthode se base sur la théorie des nombres premiers qui introduit une nouvelle approche de compresser des données temps réel.
Cette technique réduit la quantité effective à transférer à travers le réseau par conséquent minimise le temps de transmission.

Chapitre III :

Analyse et conception

1. Introduction :

Dans ce chapitre nous allons expliquer d'une manière détaillée l'analyse et la conception du système de codage /décodage de la nouvelle méthode de compression, cette dernière doit fournir en plus des fonctionnalités de compression /décompression d'autres fonctions telles que le calcul du temps et le taux de compression et de décompression.

2. Analyse et conception

Pour l'analyse et la conception on se base sur une approche modulaire, afin de rendre l'architecture plus flexible.

2.1. Quelques définitions :

a. module: Les modules sont des entités indépendantes integréés dans une architecture pour produire une application.

b. Système: L'ensemble des modules utilisés ainsi que les relations qu'ils entretiennent entre eux.

La conception consiste a élaborer a partir de la spécification du problème une solution informatique, son principe est de décomposer le problème en sous_ problème et identifier les problèmes pertinents. Nous avons deux façons de décomposer :

- Selon les fonctionnalités, traitements se qu'on appelle la *conception fonctionnelle.*
- Selon les données à manipuler et c'est *la conception objet.*

2.2. Les méthodes de conception :

On distingue deux grandes méthodes modulaires :

2.2.1. Les méthodes descendantes (Top-down) :

Qui procèdent par décomposition de problème initial en un certain nombre de sous-problèmes, chacun de complexité moindre. Cette division est ensuite appliquée aux sous problèmes générés, est ainsi de suite, jusqu'à ce que chacun des sous problèmes soit trivial et qu'on peut le résoudre par des fonctions primitives.

2.2.2. Les méthodes ascendantes (Bottom-up) :

Qui procèdent par composition de briques simples pour obtenir des systèmes complexes et ainsi de suite jusqu'as une opération globale qui résout le problème initial. C'est en particulier le cas des bibliothèques des sous programmes disponibles avec tous les systèmes, langages et environnement... etc.

3. Objectif :

Nous voulons implémenter la méthode de compression /décompression de données présentée en chapitre II cette dernière est basée sur la théorie des nombres premiers qui introduit une nouvelle manière de compresser des données, donc ce la donne la possibilité d'améliorer les performances de compression (taux, temps …)

4. Conception générale du système :

L'architecture générale de ce nouveau système de compression/décompression est composée d'un ensemble de fonctions (obtenues par la méthode descendante) qui se complètent pour réaliser l'objectif et la tâche globale visée par le système qui va être présenté dans les paragraphes qui suit :

4.1. Module principal de la méthode (le noyau) :

Afin de répondre à l'objectif, on va appliquer les concepts vue aux préalable pour définir les différents sous modules constituant le noyau, à savoir la compression, la décompression de données, le calcule et l'affichage de résultat et performances de compression/décompression.

Donc on peut décomposer le noyau en trois sous modules :

1. Module de compression ;
2. Module de décompression ;
3. Module de calcul et affichage des performances.

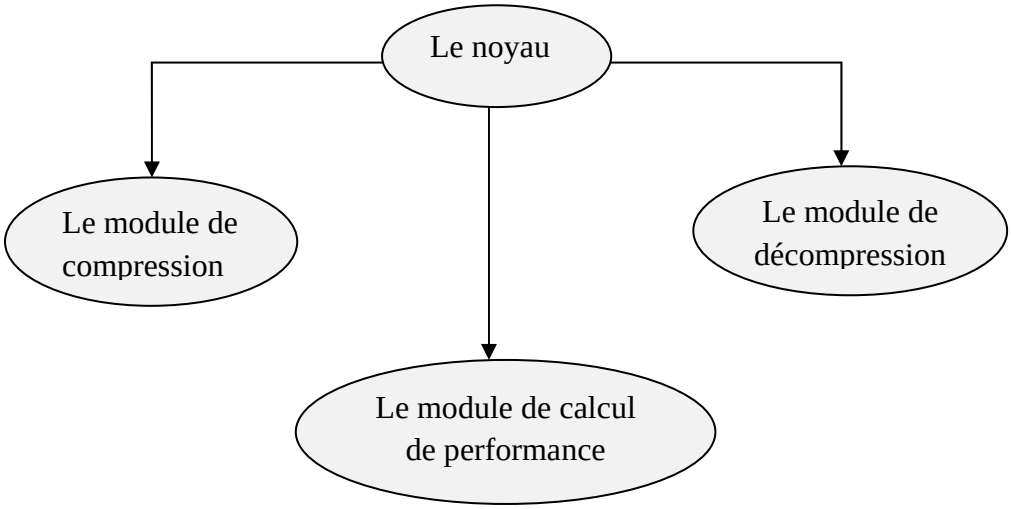


Figure III. 1 : Décomposition du module principal

4.1.1. Module de compression (encodeur) :

Avant de lancer la compression on doit d'abord choisir le fichier à compresser, le système enregistre le chemin de ce fichier. Au lancement de la compression, la fonction de compression code les données du fichier à compresser en faisant appel à la fonction **coder ( ),** qui fait la correspondance entre un caractère de la chaine de données en entrée et un nombre premier obtenu par la fonction **premiertab ( ),** comme elle fait appelle aussi a une autre fonction **ordr ()** qui retourne le code d'ordre des caractères codés précédemment, la fonction **Header_write()** est appelé pour écrire les résultats de la compression ainsi d'autres informations (taille source, nombre de symboles et la table des symboles) dans l'entête du fichier compressé .

Cette fonction transforme chaque chaine de caractère de flot de données en un enregistrement de deux attributs N_i, Ord_i tel que N_i représente le produit des nombres premiers correspondant et Ord_i permettra de conservé l'ordre des symboles.

Ainsi le module sera constitue principalement des fonctions suivantes **premiertab (), Header_write (), coder (), compresser () et ordr ()** qui se complètent pour réaliser la tâche de compression. (Voir la figure ci-dessous)

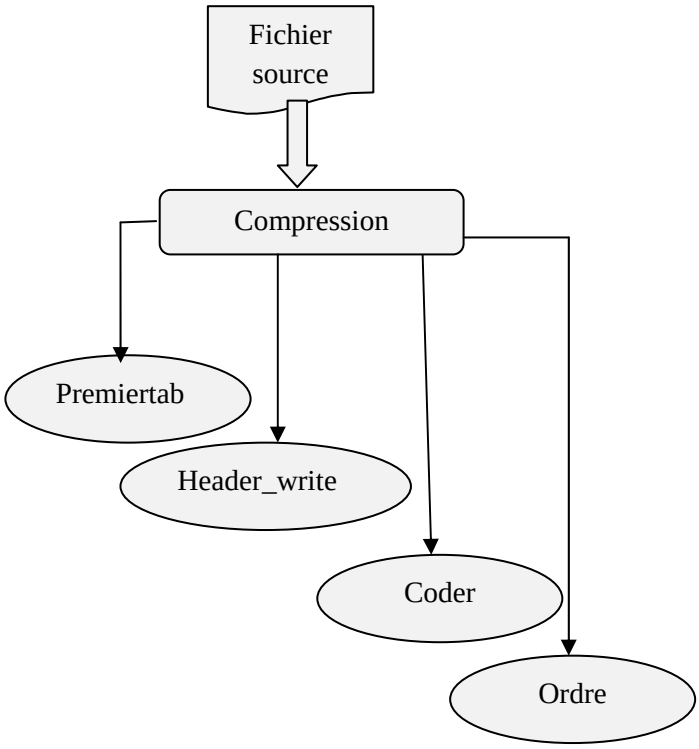


Figure III. 2 : Schéma général de module de compression

Le principe utilisé pour cette compression offre la possibilité d'affecter le plus petit nombre premier pour le symbole le plus fréquent (plus grande probabilité) et le plus grand nombre premier pour le caractère le moins fréquent.

4.1.2. Le module de décompression (décodeur):

Pour lancer la décompression, on doit d'abord choisir le fichier à décompresser et le système va enregistrer son chemin. Au lancement de la décompression, la fonction va d'abord lire les informations accompagnants le fichier compressé et qui sont enregistrer à l'entête de ce dernier en utilisant la fonction **Header_read ()**, telles que le nombre de caractères (c à d la taille de fichier) le nombre de symboles ainsi que la table des symboles. Ensuite elle va décomposer les entiers N produits par la fonction de compression en facteurs premiers et pour retrouver l'ordre des symboles dans le message elle doit passer par le processus inverse de la

fonction de codage d'ordre en se basant sur la fonction **permutation (),** puis générer l'ensemble des caractères de la chaine initiale est ce la en consultant la table de correspondance caractère /nombre premiers en utilisant la table inverse des nombres premier obtenue par la fonction **premiertab_Inv ()**

Ainsi le module sera constitue principalement de fonctions suivantes : **Header_read (), premiertab_Inv(), permutation() et décompression (),** qui se complètent pour réaliser la tâche de décompression.(voir la figure).

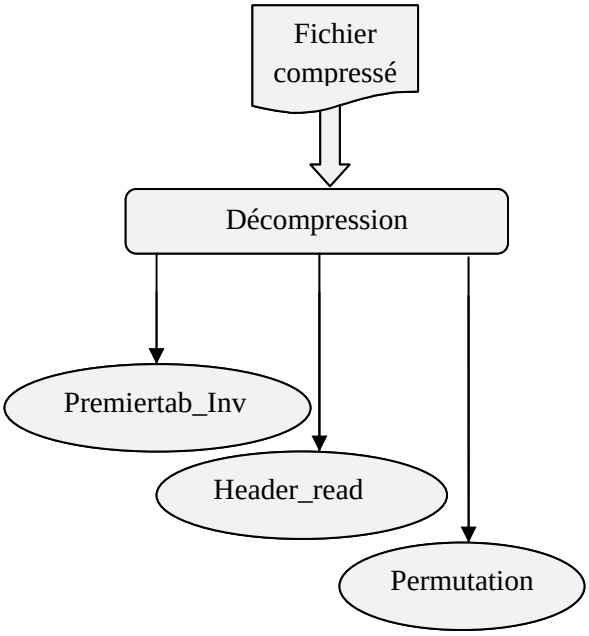


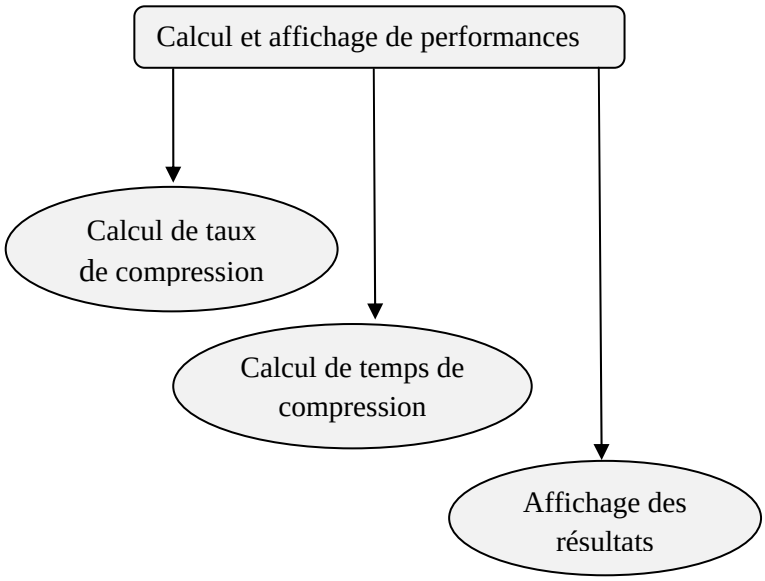
Figure III. 3 : Schéma général de module de décompression

Le principe consiste à retrouver le même fichier source sans aucune perte de données.

4.1.3. Le module de calcul et affichage de performance :

Ce module calcul et affiche les performances de l'algorithme comme le temps de compression /décompression, et le taux de compression. C'est grâce à celui-ci que nous allons tester et évaluer cette méthode de compression, donc il peut être considérer comme la partie « test et évaluation de l'algorithme ».

a. calcul et affichage des performances pour la compression :



FigureIII. 4 : Schéma général du module du calcul et affichage des performances pour la compression

b. calcul et affichage des performances pour la décompression :

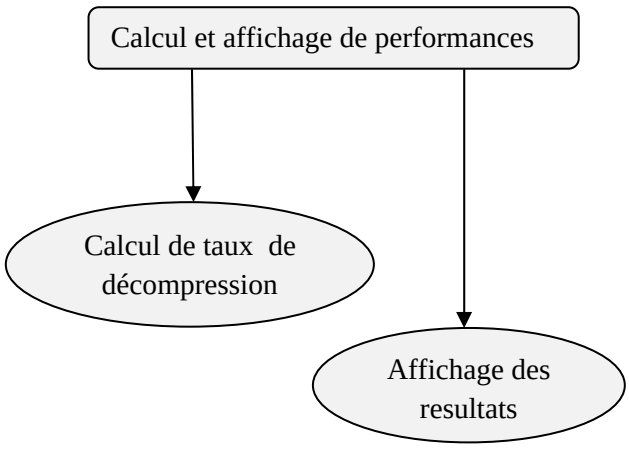


Figure III. ? : Schéma général du calcul et affichage des performances pour la décompression

4.1.3. 1. Le taux de compression

Pour calculer le taux de compression le module de calcul de performance doit d'abord reconnaitre en entrée et calculer la taille de fichier à compresser (source) et le fichier compressé (cible), puis il calcule le taux selon la formule suivante :

$$T=1 - \frac{Taille\ compressée}{Taille\ source} \times 100$$

4.1.3. 2. Le temps de compression

Pour calculer le temps de compression ce module utilise ou sollicite l'horloge système et cela par l'appelle de la fonction clock () avant le début de la compression pour lui donner le temps de début de la compression, à la fin de la compression il sollicite une autre fois la fonction clock () pour lui donner le temps de fin de la compression

Pour obtenir le temps de compression exacte, il fait la soustraction du temps fin de la compression par le temps début de la compression.

Temps de compression = temps fin de compression – temps début de compression

4.1.3..3. Le temps de décompression

Le temps de décompression est calculé de la même manière que celui de la compression en sollicitant l'horloge système en utilisant la fonction clock () au début et fin de décompression pour obtenir respectivement le temps de début et fin de décompression.

Temps de décompression = temps fin de décompression – temps début de décompression

4.2. Les algorithmes :

On présente les algorithmes des modules essentiels qui forment le noyau de l'application tel que la compression et la décompression.

4.2.1. Algorithme de compression

DEBUT

1. Ouvrir le fichier à compresser ;

2. Créer un nouveau fichier pour sauvegarder les résultats de la compression ;

Faire

3. Recevoir les caractères de flot de données d'entrée et attribuer au code ASCII de chaque caractère le nombre premier correspondant selon sa probabilité d'apparition dans le fichier source pris de la table de correspondance caractère/nombre premier ;

4. Calculer l'attribut Ordre qui permet de reconstitué l'ordre des symboles dans le message à l'aide de la fonction ordr () ;

5. Sauvegarder le résultat dans le fichier crée ;

Tantque (il y'a de caractère) **;**

6. Fermer les deux fichiers source et compressé.

FIN

4.2.2. Organigramme de compression :

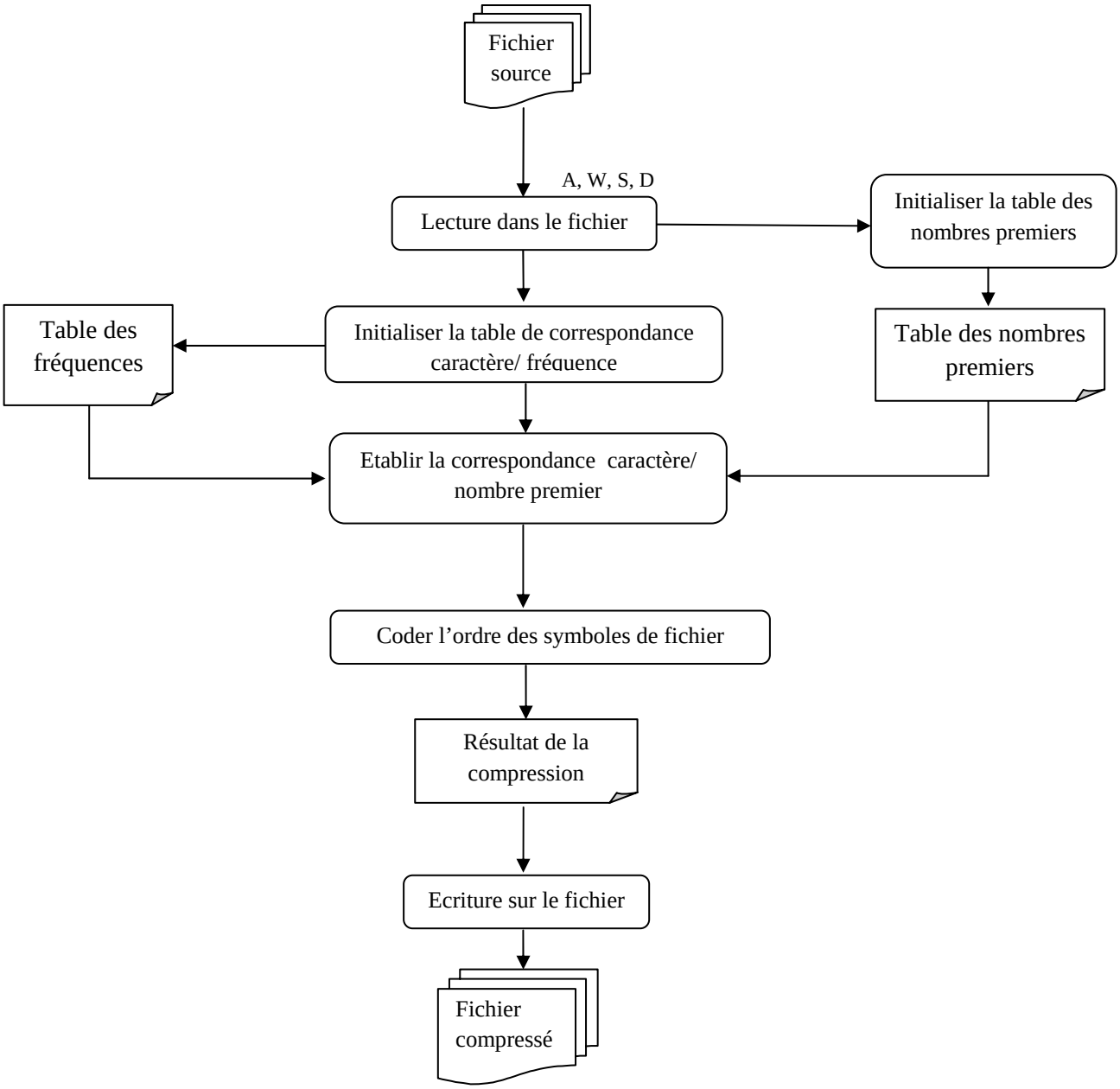


Figure III. 5 : Illustration du module de compression

4.2.3. Algorithme de décompression

DEBUT

1. Ouvrir le fichier compressé (à décompresser) ;

2. Créer un nouveau fichier pour sauvegarder les résultats de la décompression ;

3. Charger l'entête du fichier compressé ;

Faire

2. Décomposition d'entiers en ses facteurs premiers on se basant sur la fonction Décomposition () et pour retrouver l'ordre des caractères on doit effectuer le processus inverse de la fonction de codage de l'ordre.

Faire

3. Chercher le caractère correspondant à chaque facteur dans la table de correspondance en utilisant la table inverse des nombres premiers.

Tantque (Il reste des entiers) **;**

Tantque (on n'a pas atteint le nombre de caractères spécifies dans l'entête) **;**

4. fermer les deux fichiers ouverts précédemment.

FIN

4.2.4. Organigramme de décompression

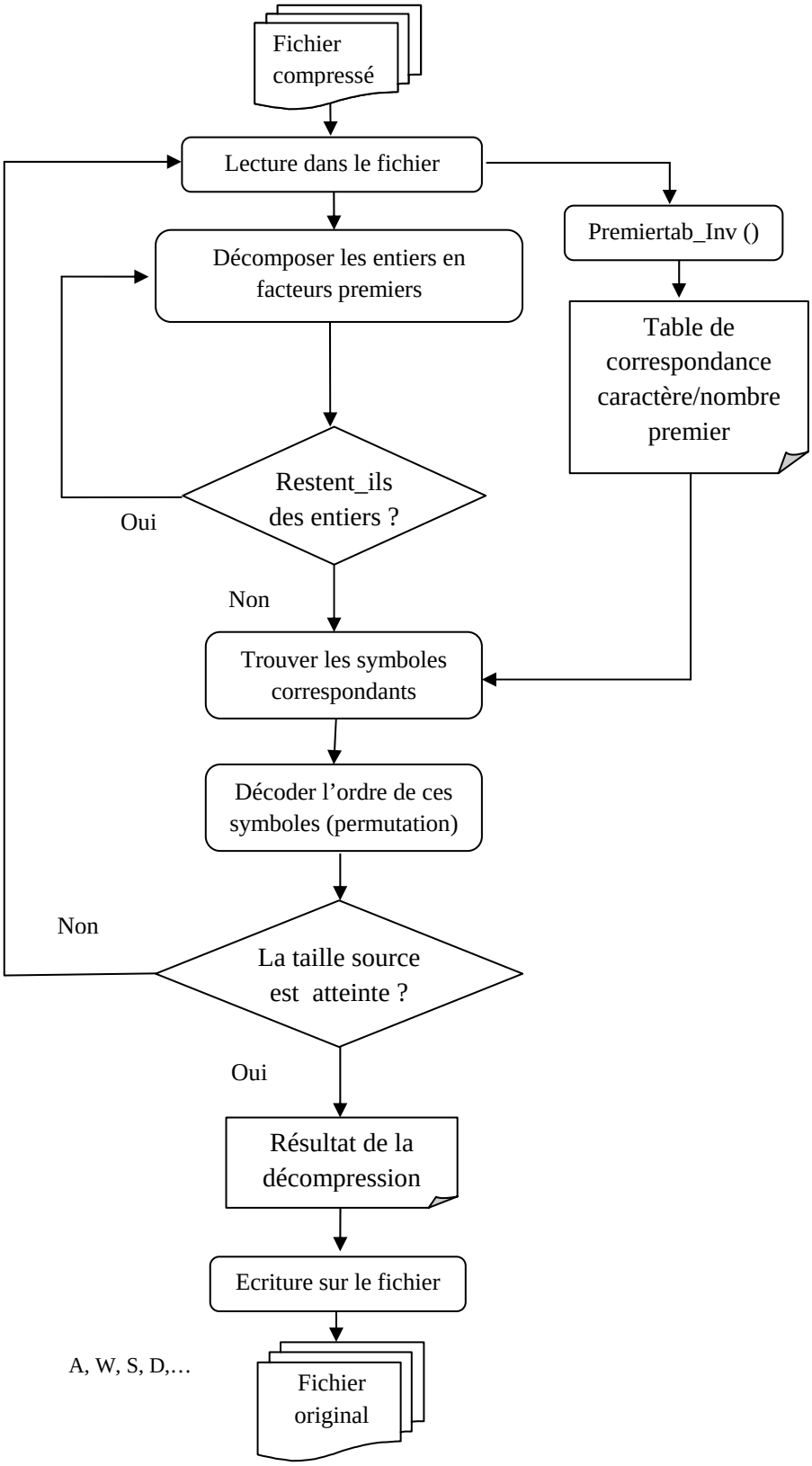


Figure III. 6 : Illustration du module décompression

4.2.5. Algorithme de calcul du taux de compression

DEBUT

1. Récupérer la taille de fichier source Taille_src
2. Calculer la taille de fichier compressé Taille_cmp
3. Taux= ((1-Taille_cmp/Taille_src)*100)

FIN

4.2.6. Algorithme de calcul du temps de compression/décompression

a. Compression

DEBUT

1. Déb = clock ()
2. Compression
3. Fin = clock ()
4. Temps= (Fin-Déb)

FIN

a. Décompression

DEBUT

1. Déb = clock ()
2. Décompression
3. Fin = clock ()
4. Temps= (Fin-Déb)

FIN

Pour donner une idée générale et bien illustrée du fonctionnement de cet algorithme, le schéma ci-dessous représente la compression /décompression, calcul et affichage des performances.

4.2.7. Organigramme général :

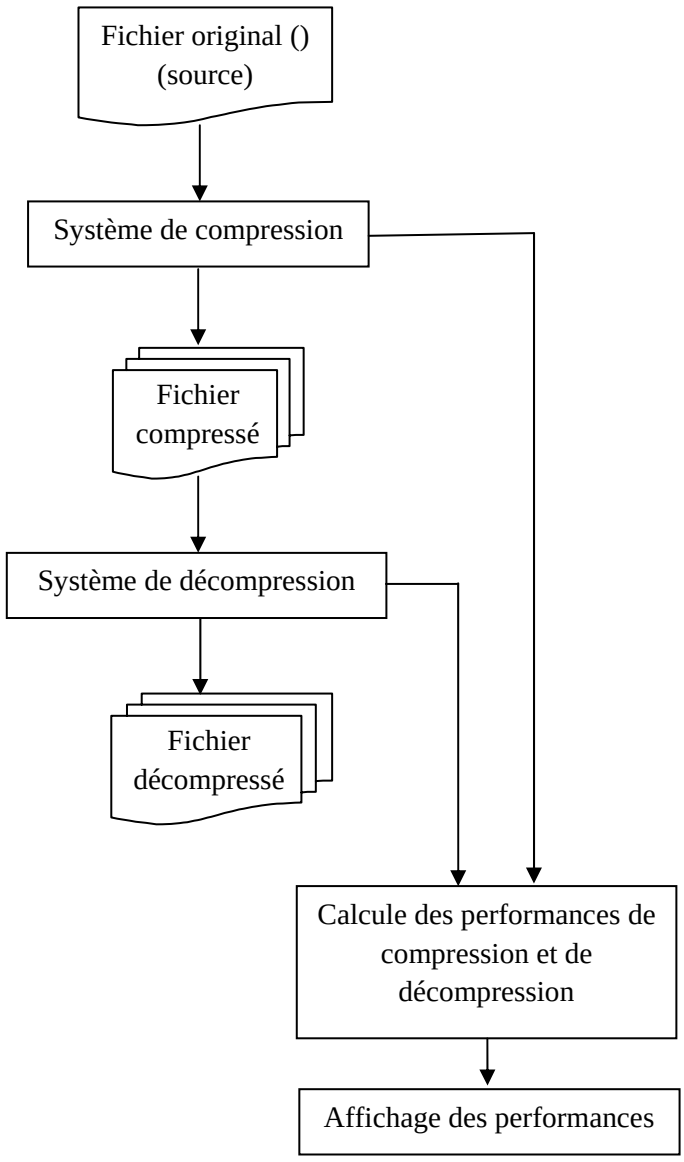


Figure III.7 : Schéma général de la methode

5. Conclusion

Dans ce chapitre nous avons spécifie les stratégies qui devrons mise en œuvre, pour atteindre l'objectif de notre travail. Ici nous avons fait l'analyse et la conception de la nouvelle méthode de compression, et nous l'avons considérée comme étant un problème (noyau) et nous l'avons décomposée en sous problèmes (modules) et chaque module en sous modules et nous avons illustré tous ça avec des schémas.

En fin avec ce formalisme assez important, nous sommes en mesure d'aborder l'implémentation qui mettre en pratique tout ce qui a été présenté dans cette partie, en détaillant d'avantage les modules et la manière de les implémenter.

Chapitre IV :

Tests et implémentation

1. Introduction :

Dans le chapitre précédent, nous avons parlé de la conception de notre nouvelle méthode de compression. A présent, nous allons détailler son implémentation ainsi que le résultat des différents tests effectués dans le but d'évaluer les performances. A chaque fin d'un test, nous allons comparer les résultats obtenus entre la méthode proposée et une autre méthode que nous choisirons par la suite.

2. Environnement de développement :

a. Système d'exploitation :

Le système d'exploitation Windows est un environnement graphique organisé en fenêtres. Il offre à l'utilisateur une interface graphique multifenêtrage et une gestion multitâches, qui facilite le travail des développeurs.

b. Langage de programmation:

- **Un peu d'histoire [10] :**

Le langage C++ a deux grands ancêtres :
– Simula, dont la première version a été conçue en 1967. C'est le premier langage qui introduit les principaux concepts de la programmation objet. Probablement parce qu'il était en avance sur son temps, il n'a pas connu à l'époque le succès qu'il aurait mérité, mais il a eu cependant une influence considérable sur l'évolution de la programmation objet.
– Le langage C a été conçu en 1972 aux laboratoires *Bell Labs*. C'est un langage structuré et modulaire, dans la philosophie générale de la famille Algol. Mais c'est aussi un langage proche du système, qui a notamment permis l'écriture et le portage du système Unix. Par conséquent, la programmation orientée système s'effectue de manière particulièrement aisée en C, et on peut en particulier accéder directement aux fonctionnalités du noyau Unix.
Le concepteur de C++, Bjarne Stroustrup, qui travaillait également aux *Bell Labs*, désirait ajouter au langage C les classes de Simula. Après plusieurs versions préliminaires, le langage a trouvé une première forme stable en 1983, et a très rapidement connu un vif succès dans le monde industriel. Mais ce n'est qu'assez récemment que le langage a trouvé sa forme définitive, confirmée par une norme.

C++ peut être considéré comme un successeur de C. Tout en gardant les points forts de ce langage, il corrige certains points faibles et permet l'abstraction de données. De plus, il permet la programmation objet.

- **Définition [9] :**

Le C++ est l'un des langages de programmation les plus utilisés actuellement. Il est à la fois facile à utiliser et très efficace. Il souffre cependant de la réputation d'être compliqué et illisible. Cette réputation est en partie justifiée. La complexité du langage est inévitable lorsqu'on cherche à avoir beaucoup de fonctionnalités. En revanche, en ce qui concerne la lisibilité des programmes, tout dépend de la bonne volonté du programmeur.
Les caractéristiques du C++ en font un langage idéal pour certains types de projets. Il est incontournable dans la réalisation des grands programmes. Les optimisations des compilateurs actuels en font également un langage de prédilection pour ceux qui recherchent les performances. Enfin, ce langage est, avec le C, idéal pour ceux qui doivent assurer la portabilité de leurs programmes au niveau des fichiers sources (pas des exécutables).

- **Avantages et inconvénients [8] :**

Les principaux avantages du C++ sont :
– Il offre un plus grand nombre de fonctionnalité,
– Il profite des fonctionnalités et de la performance du langage C, lequel est incontournable dans la programmation système,
– Il permet de raisonner et de programmer en orienté objet,
– Il permet une portabilité des fichiers sources (effort de standardisation),
– Il possède un système de contrôle d'erreurs important.
Tous ces avantages ont fait que C++ est le langage le plus utilisé actuellement. On peut citer pour preuve de l'engouement qu'il suscite dans le domaine de programmation, le nombre important d'implémentations (Compilateurs) et environnements graphiques mis sur le marché (Borland C++, Borland C++ builder, Microsoft Visual C++, fltk, Qt designer, Kdevelop, g++, ...).
L'inconvénient du langage C++ est essentiellement sa complexité d'apprentissage. En effet, les multiples fonctionnalités offertes mais aussi les différentes façons de penser un problème font que l'apprentissage de l'utilisation efficace du C++ nécessite plus que la simple

connaissance d'un ensemble de constructions syntaxiques et la sémantique correspondante. Les exemples réels et pratiques sont d'un apport considérable dans ce processus d'apprentissage.

c. L'environnement de développement [12]:

Comme environnement de développement on a choisi le Dev C++ qui est un IDE (Integrated Development Environment) 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 et est régulièrement mis à jour.La capture d'écran suivante c'est la version 4.9.9.2

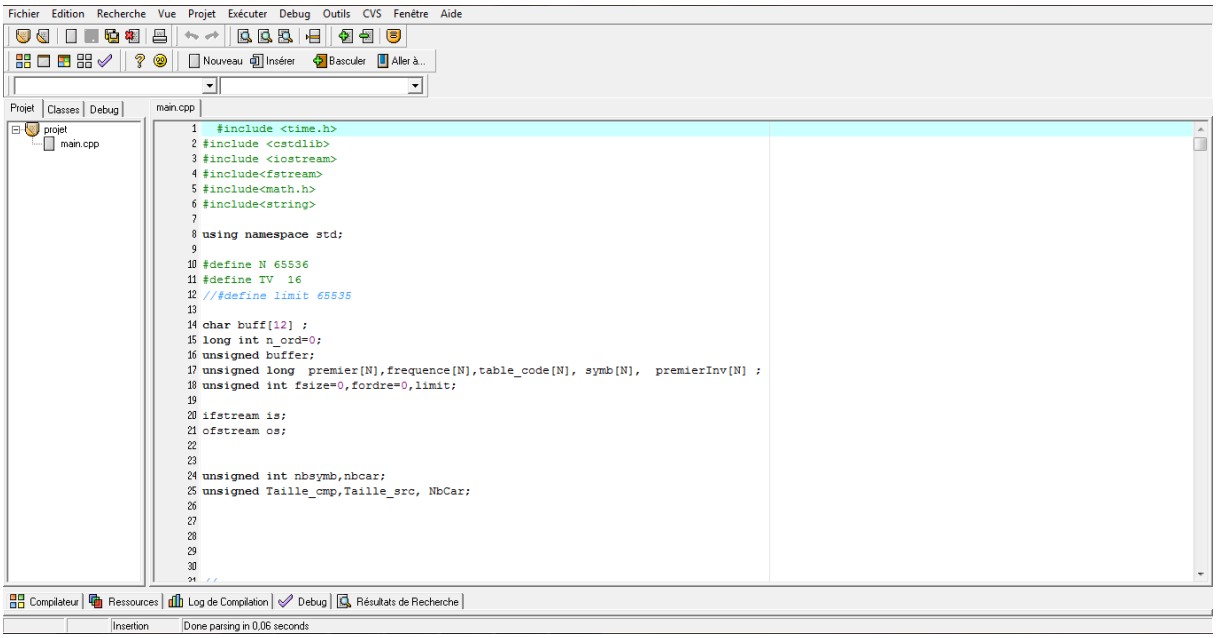


Figure VI. 1: Dev C++ 4.9.9.2

3. Le noyau de la méthode

Le noyau de cette méthode est constitué de plusieurs fonction qui se complètent pour répondre à l'objectif visé.les fonctions principales sont les suivantes :

- void premiertab () : Cette fonction est utilisée pour initialiser le table des nombres premiers.
- int lire () : lit le fichier d'entrée caractère par caractère est calcule leur fréquences.

- void coder () : Fait la correspondance entre chaque caractère du fichier d'entrée et les nombres premiers en respectant le principe suivant : affectation de plus petit nombre premier au caractère le plus fréquent.
- void Header_write() : cette fonction est appelée avant la compression pour écrire l'entete du fichier compressé.
- void Compression () : elle s'occupe de la lecture du fichier source et l'écriture dans le fichier cible. Elle fait appelle à d'autres fonctions telles que int ordr (),void write_ordre () , void write_val () et bien d'autres qui s'accomplissent pour effectuer le codage.
- void Header_read () : elle est appelée pour lire l'entête du fichier compressé avant d'effectuer la décompression.
- void premiertab_Inv() : son rôle est de remplir la table inverse de la table des nombres premiers.elle est utilisée lors de la décompression.
- void Decompression () : elle parcourt le fichier compressé, lit l'entête qui l'accompagne et appelle d'autre fonction telles que unsigned int ReadVal(int taille), void trier (char buff[], int n), unsigned int ReadOrd(int taille) et

void permutation (unsigned int code, int taille) qui se complètent entre elles et écrit dans le fichier décompressé.

- void permutation (unsigned int code, int taille) : cette fonction a le rôle de retourner sous une chaine de caractère le code de l'entier « code » tout en connaissant l'entier « taille » à la fonction de décompression.

4. Evaluation et comparaison :

Pour faire la comparaison on choisit une autre méthode c'est ***« la méthode de Huffman »***

4.1. Evaluation du taux de compression :

Nous allons faire cette comparaison par rapport à deux critères : le ***type*** et aussi la ***taille*** de fichier. Dans cette comparaison, nous avons pris pour chaque type de fichier (texte, image, son) un ensemble de fichier de même type mais de taille différente et nous avons fait les tests sur chacun d'eux.

Ces différents fichiers ont été téléchargés de *corpus de Calgary* [1] pour comparer les algorithmes de compression qui étaient présentés. Le corpus élaboré avant 1990 est constitue

de fichiers de tailles variables, de natures différentes, ces fichiers ont été choisi de façon à être représentatifs des fichiers que nous étions susceptibles de rencontrer.

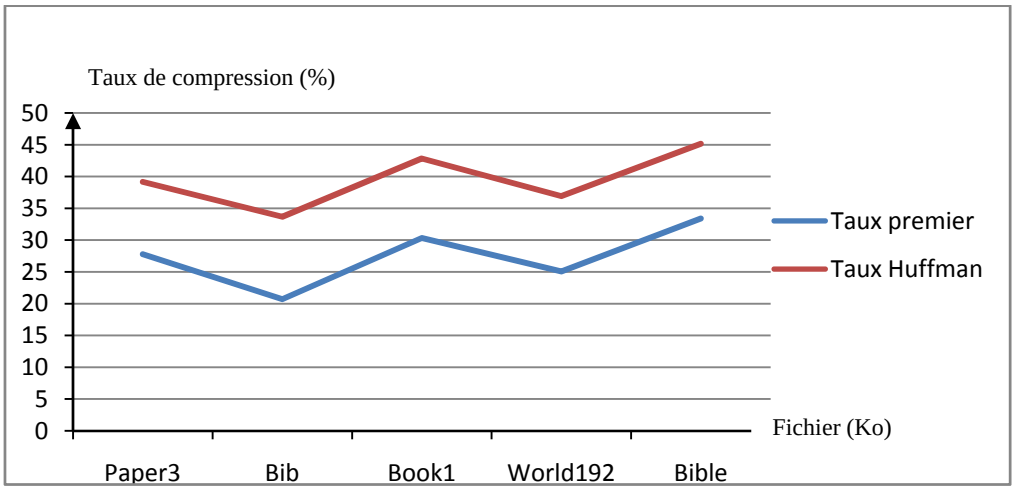
- **Les fichiers « texte » :**

  Nous commençons par les fichiers texte pour cela nous avons pris un ensemble de fichiers textes de tailles différentes :

Le nom de fichier	La taille de fichier (Ko)
Paper3	46
Bib	109
Book1	751
World192	2416
Bible	3953

Tableau VI. 1 : Les tailles des fichiers texte utilisés pour le test d'évaluation

Les résultats obtenus sont représentés sur la figure suivante :



FigureVI. 2 : Evaluation du taux de compression pour les fichiers texte

A partir de la figure nous remarquons que le taux de compression est meilleur avec la méthode de Huffman qu'avec celle basée sur les nombres premiers.

- **Les fichiers « image » :**

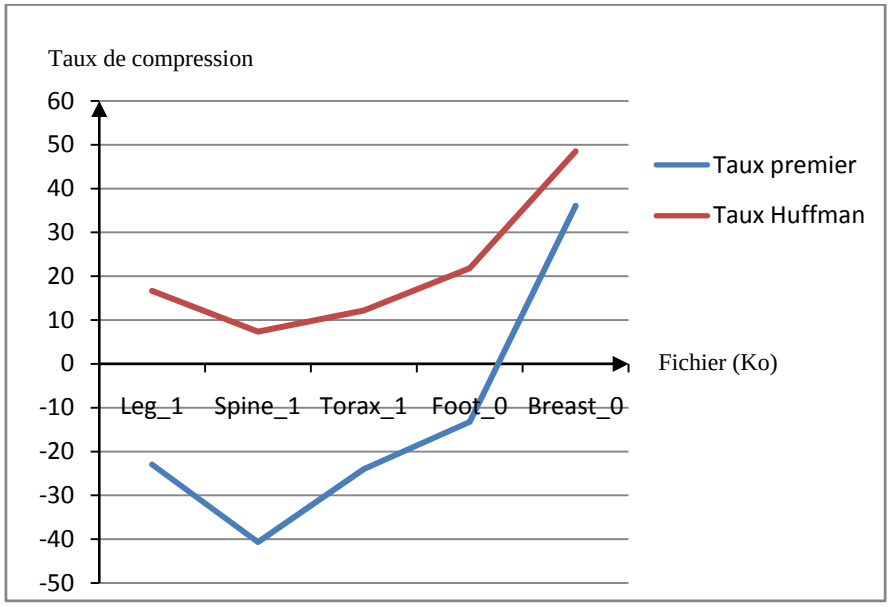
Pour l'évaluation de taux de compression pour les fichiers image on a pris une collection d'image de deux formats différents (tif, dicom), avec des tailles différentes ;

1.

Le nom de fichier	La taille de fichier (Ko)
lukas_2d_8_leg_1_t.tif	1282
lukas_2d_8_spine_1_t.tif	2190
lukas_2d_8_thorax_1_t.tif	2788
lukas_2d_8_foot_0_t.tif	3052
lukas_2d_8_breast_0_t.tif	3665

Tableau VI. 2 : Les tailles des fichiers image (.tif) utilisés pour le test d'évaluation

Les résultats obtenus sont représentés sur la figure suivante :



FigureVI. 3 : Evaluation du taux de compression pour les fichiers image (.tif)

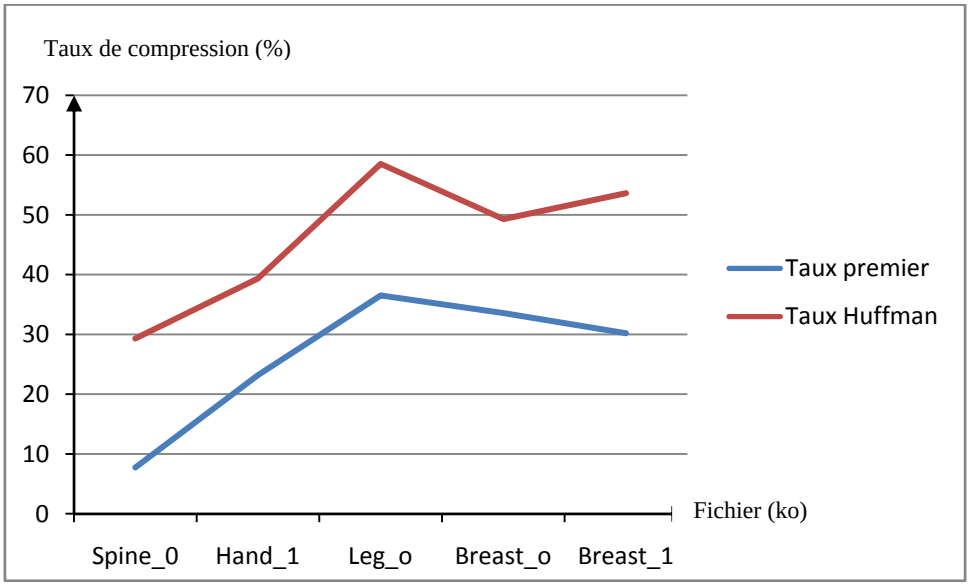
On remarque que le taux de compression pour la méthode de Huffman est meilleur contrairement à la méthode proposée qui représente des taux négatifs.

2.

Le nom de fichier	La taille de fichier (Ko)
lukas_2d_16_breast_0.dcm	3680
lukas_2d_16_hand_1.dcm	6085
lukas_2d_16_leg_0.dcm	7359
lukas_2d_16_breast_0.dcm	8195
lukas_2d_16_breast_1.dcm	8195

Tableau VI. 3: Les tailles des fichiers image (.dcm) utilisés pour le test d'évaluation

Les résultats obtenus sont représentés sur la figure suivante :



FigureVI. 4 : Evaluation du taux de compression pour les fichiers image (.dcm)

A partir de la figure précédente on remarque que pour ce type d'image le taux de compression de la méthode de Huffman est visiblement meilleur que celui de la méthode de proposée.

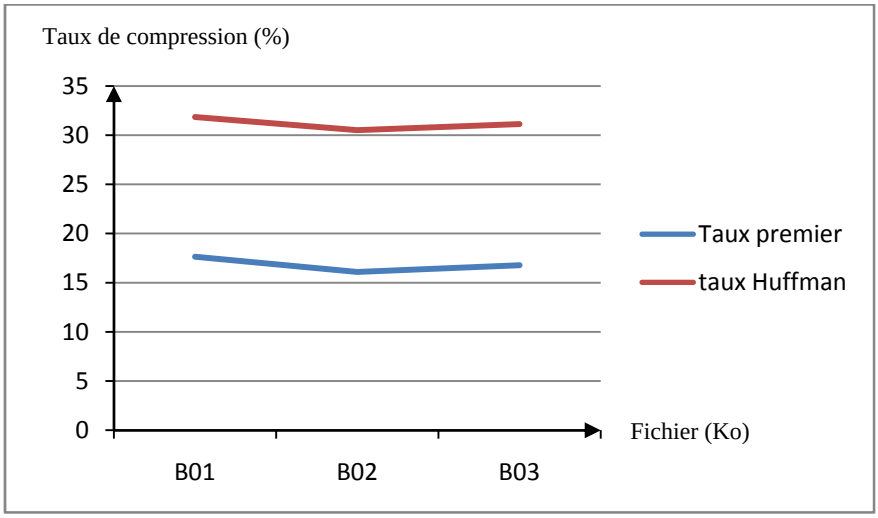
- **Les fichiers « web» :**

Pour l'évaluation de taux de compression pour les fichiers web on a pris un ensemble des pages web (d'extension .HTML), avec des tailles différentes.

Le nom de fichier	La taille de fichier (Ko)
B01	2076
B02	2084
B03	2123

Tableau VI. 4: Les tailles des fichiers web utilisés pour le test d'évaluation

Les résultats obtenus sont représentés sur la figure suivante :



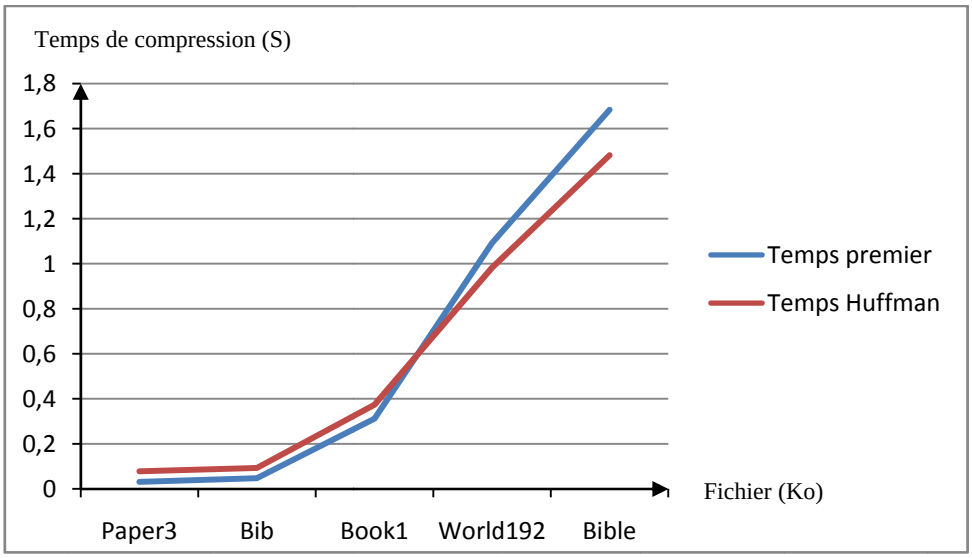
FigureVI. 5 : Evaluation de taux de compression pour les fichiers web

Le taux de compression de la méthode de Huffman est beaucoup plus meilleur avec ce type de fichier comme le montre la figure précédente.

4.2. Evaluation du temps de compression :

Pour évaluer le temps de compression, on va comparer les deux méthodes par rapport à leurs temps d'exécution. Les valeurs du temps représentés dans les figures qui suivent sont celles calculées à partir des tests précédents pour calculer le taux de compression.

- **Les fichiers « texte » :**

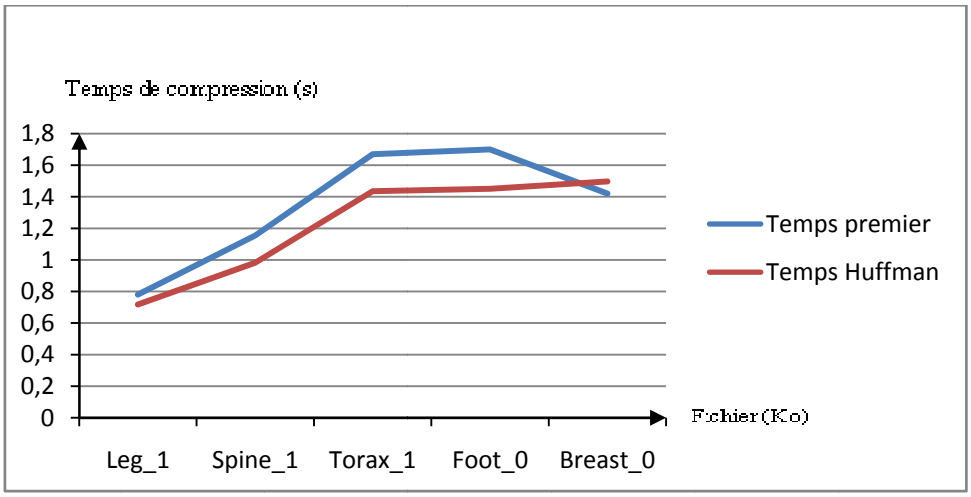


FigureVI. 6: Evaluation du temps de compression pour les fichiers texte

Le temps de compression pour les fichiers texte est presque le même pour les deux méthodes, on remarque que le temps de compression est proportionnel à la taille de fichier ce qui apparait évident.

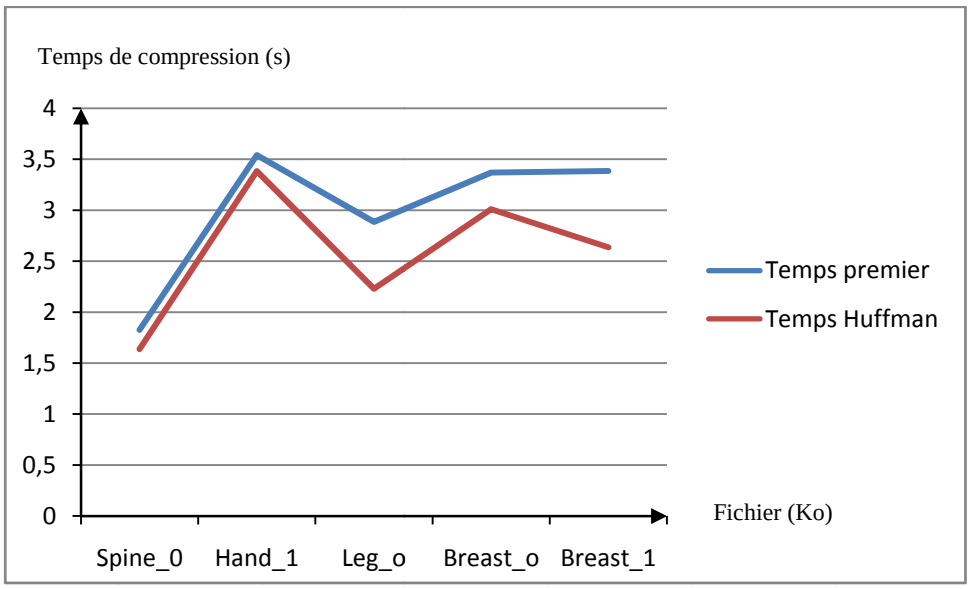
- **Les fichiers « image » :**

1. Avec extention.tif



FigureVI. 7 : Evaluation du temps de compression pour les fichiers image

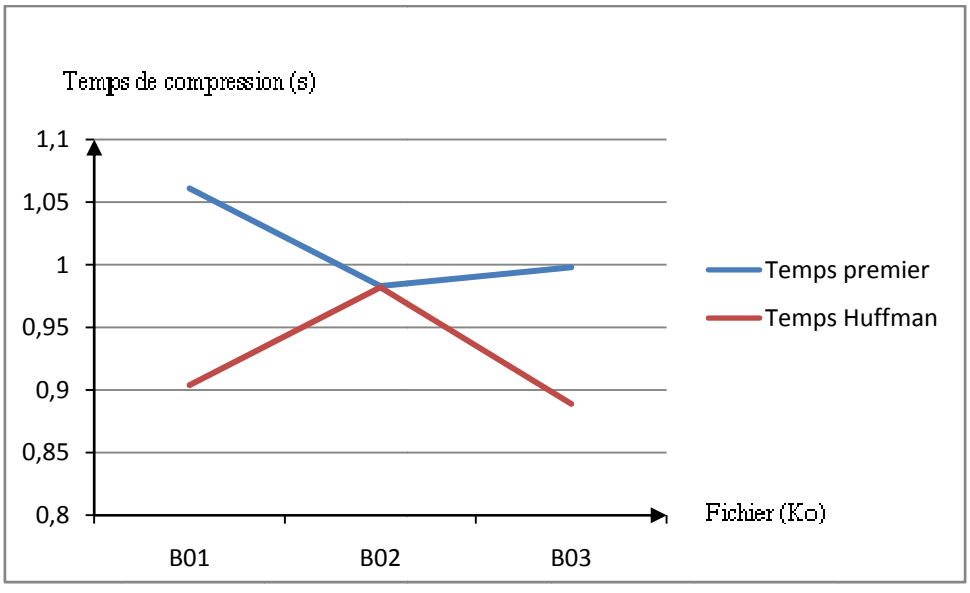
2. Avec extention.dcm



FigureVI. 8 : Evaluation du temps de compression pour les fichiers image

Pour les deux figures précédentes le temps de compression de la méthode de Huffman est inferieur par rapport au temps de compression avec la méthode basée sur les nombres premiers.

- **Les fichiers « web» :**

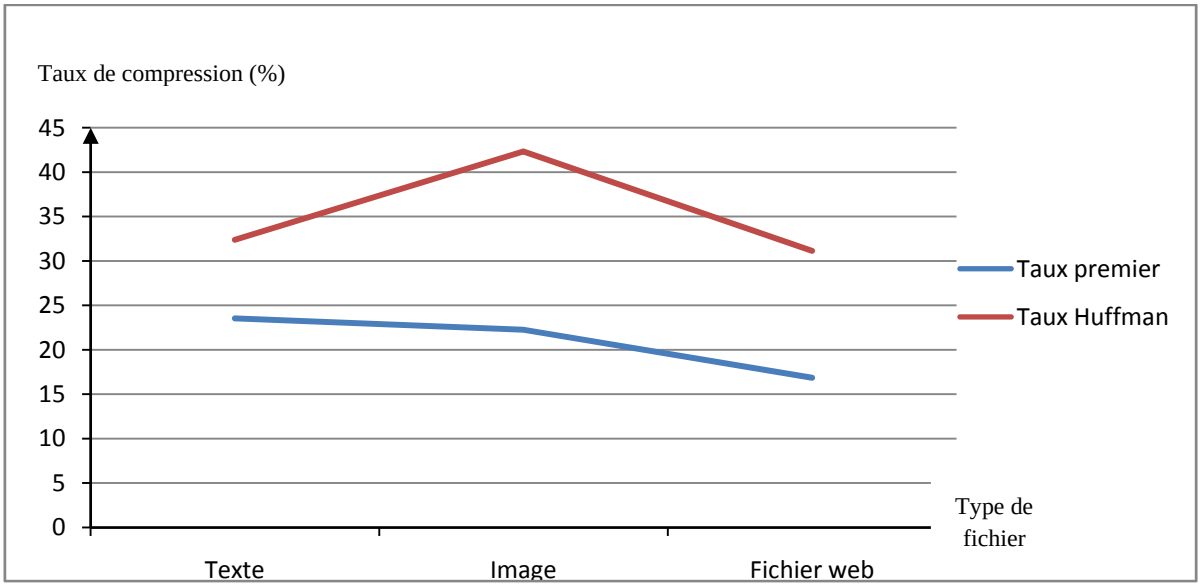


FigureVI. 9 : Evaluation du temps de compression pour les fichiers web

On remarque que le temps de compression pour la méthode proposée est beaucoup plus élevé que celui de la méthode de Huffman.

4.3. Evaluation du taux de compression pour les différents types de fichier:

Pour cela on calcule la moyenne des taux pour chaque type de fichier et on obtiendra le graphe suivant :



FigureVI. 10 : Evaluation du taux de compression pour les différents types fichiers

D'après la figure, le taux de compression est meilleur avec la méthode de Huffman pour les differents types de fichiers que pour la méthode proposée.

5. Interpretation :

Aprés avoir vu les résultats obtenus des différents tests d'évaluation effectués sur les deux méthodes de compression, on interprétera les résultats obtenus dans ce qui suit :

- Taux de compression : Nous avons une meilleur compression avec la méthode de Huffman avec les différents échantillons des différents types de fichier que nous avons choisi qu'avec la méthode proposée et ca revient au nombre limité des symboles codés qui est du à la fonction de multiplication utilisée qui rend les nombres naturels, résultant de la composition, très grand ce qui limite le taux de compression.

- Temps de compression : le temps de compression de la méthode proposée est un peu élevé par rapport à celui de Huffman et il est plus remarquable avec les fichiers web cela est du au temps considérable qu'on accorde au calcul des entiers correspondant aux différents symboles qui apparaissent dans le fichier et au codage d'ordre de ces derniers (complexité algorithmique de la méthode proposée).

6. Perspectives :

La méthode basée sur les nombres premiers est appelés à être améliorée, nous proposons :

- Augmenter le nombre de symbole à codé
- Etendre la table de nombre premier a une taille >>N
- Simplifier l'algorithme de la méthode …

7. Conclusion :

Ce chapitre présente en premier lieu l'environnement du travail ainsi les principales fonctions de la méthode proposée. En deuxième lieu nous avons donné les résultats obtenus après les tests effectués (le taux et le temps par rapport aux types et taille du fichier) avec les interprétations des schémas.

Conclusion Générale

Conclusion générale

Les différentes lacunes causées par la redondance des messages à stocker ou à transmettre a poussé plusieurs concepteurs à définir différents algorithmes de compression de donnée, afin d'exploiter au maximum les ressources de stockage et de transmission. Les systèmes actuels procèdent ainsi à l'encodage des messages faisant appel au plus petit nombre de bits possibles, pour une meilleure optimisation.

Dans ce mémoire une nouvelle méthode de compression de données qui repose sur un modèle mathématique (théorie des nombres premiers) a été présenté . Pour situer le travail, nous avons exposé les outils nécessaires à l'approche de compression par une présentation de quelques généralités, définitions et notions de base sur le domaine d'étude.

L'étude faite nous a permis d'acquérir beaucoup de connaissance dans un domaine aussi vaste qui est la compression de données, ce travail a été aussi bénéfique en termes de programmation C++.

En fin nous espérons avoir parvenue à répondre à l'objectif fixé initialement à savoir, l'implémentation d'une méthode de compression /décompression de données basée sur la théorie des nombres premiers.

Bibliographie

Bibliographie

[1] Steven Pigeon : contribution à la compression de données, Décembre 2001.

[2] Pereira Vincent - LEPRETTE Franck - HACAULT Vincent : Compression de données. Décembre 2004.

[3] Markus Jaton : Compression de fichiers

[4] S.Maadi, Y. Peneveyre, et C. Lambercy : Compression de données sans pertes

[5] http://www.wikipédia.org

[6] http://www.revue-eti.net/

[7] http://www.techno-science.net

[8] Outouati Sonia : C++ : Le cours Informatique

[9] Christian Casteyde : Cours de C/C++

[10] Karl Tombre ; École des Mines de Nancy : Une courte introduction à C++ .*Version 1.0.* Octobre 1999

[11] http://www.Siteduzero.com

[12] Henri Garreta : Utiliser Dev-C++ : Faculté des Sciences de Luminy

[13] MM. S.Maadi, Y. Peneveyre, et C. Lambercy ‘’Compression de données avec pertes’’.

[14] David Salomon ‘’Data Compression The Complete Reference Fourth Edition’’.

Email: david.salomon@csun.edu