République Algérienne Démocratique et Populaire
Ministère de l'Enseignement Supérieur et de La Recherche Scientifique
Université Mouloud MAMMERI de Tizi-Ouzou
Faculté de Génie électrique et Informatique
Département Informatique

MEMOIRE

de fin d'études

En vue de l'obtention de diplôme master en informatique

Option : système informatique

Thème

Implémentation et évaluation d'une méthode de compression de données texte par graphe acyclique (arbre)

Réalisé par :

M^r^ Ait Yakoub Ahcene

M^r^ Ait Tayeb Sofiane

Proposé et dirigé par :

M^r^ S.Sadou

Promotion : 2014/2015

Remerciements

Nous remercions dieu le tout puissant pour nous avoir donné le courage et la volonté pour la réalisation de ce travail.

Nos plus vifs remerciements à Monsieur S.Sadou pour nous avoir proposé ce thème. Nous lui somme très reconnaissants pour ses remarques pertinentes, ses conseils judicieux et sa disponibilité à nous écouter à tout moment.

Nous remercions également les membres du jury, devant qui nous avons l'honneur d'exposer notre travail, et qui ont pris la peine de lire avec soin, ce mémoire pour juger son contenu.

Nos remerciements, vont aussi à tous ceux qui nous ont encouragées, à la réalisation de notre modeste travail.

Dédicaces

Je dédie ce travail :

A mes très chers parents.

A mes frères et sœurs.

A tous mes amis

Mon binôme Sofiane

Je remercie tous ceux et celles qui m'ont aidé à réaliser ce travail.

AHCENE

Dédicaces

Je dédie ce travail :

A mes très chers parents.

A mes sœurs.

A tous mes amis

Mon binôme Ahcene

Je remercie tous ceux et celles qui m'ont aidé à réaliser ce travail.

SOFIANE

Sommaire

CHAPITRE II

Compression par arbre

CHAPITRE III

Analyse te conception

CHAPITRE IV

Implémentation et évaluation

Annexes

Liste des figures

Chapitre I

Chapitre II

Chapitre III

Chapitre IV

Introduction générale

Au début de l'air informatique, les ordinateurs disposaient de très peu de mémoires, aussi bien vive que morte, avec des disques durs de très faible capacité.

Le stockage et la transmission des données ont toujours constitué d'une façon économique un problème de taille à cause de la limite des capacités des supports de stockage et du débit des canaux de transmission. Aujourd'hui, les équipements informatiques sont largement plus performants qu'avant, un ordinateur moyenne gamme dispose de 1 Giga-octets de mémoire vive, 100 Giga-octets de disque dur et un processeur d'au moins 1GHz de fréquence.

Cependant malgré cette ascension de la technologie des ordinateurs, les données traitées demandent de plus en plus d'espace, on peut citer à titre d'exemple les fichiers images ou vidéos.

La compression de données est considérée comme une solution qui peut être employé pour alléger une portion du problème du stockage et de transfert des données. Le gain en espace de stockage obtenu à travers la compression n'est pas avantageux juste pour le disque dur, mais aussi au niveau de la transmission par réseau. En effet plus les données occupent moins d'espace, plus la transmission est rapide.

L'objectif de notre travail est d'implémenter et d'évaluer une méthode de compression basée sur les arbres, qui permettra la réduction de l'espace occupé par les données texte.

Et pour bien mener notre travail, nous avons adopté la structure suivante :

- **Le premier chapitre** est une introduction générale à la compression de données, dans lequel nous présenterons les définitions essentielles et les différents concepts fondamentaux de la compression de données.
- **Le second chapitre** a pour objectif de présenter quelques approches de compression de données basées sur les arbres, ainsi que quelques algorithmes et techniques de compression.
- **Le troisième chapitre** a pour objectif, d'une part, de présenté l'application, et d'autre part, de décrire d'une façon détaillée l'analyse et conception de la démarche adoptée pour la réalisation de notre application.
- **Le quatrième et dernier chapitre** a pour but de présenté les différentes fonctions et outils nécessaires à la réalisation de notre travail, suivi d'une évaluation de l'application.

Enfin nous clôturerons notre travail par une conclusion générale.

Chapitre I

Généralités sur la compression de données

I)-Introduction

La compression de données, de façon simplifiée, est l'ensemble des méthodes que l'on utilise pour prendre un message long, et en faire un message court, sans perdre d'information importante. Nous trouvons de tels usages dans la vie de tous les jours et pas seulement dans les applications technologiques. Nous utilisons constamment des abréviations pour prendre des notes manuscrites plus rapidement, nous remplaçons des noms d'items par leurs numéros d'inventaire, etc.

II)-Historique sur la compression de données [17]

Le code Morse, inventé en 1838 pour une utilisation dans la télégraphie, est un exemple précoce de compression de données basée sur l'utilisation de code plus courts pour les lettres telles que "e" et "t" qui sont plus fréquentes en anglais. Le travail moderne sur la compression de données a commencé dans les années 1940 avec le développement de la théorie de l'information. En 1949, Claude Shannon et Robert Fano, ont conçus de façon systématique pour attribuer des codes basés sur les probabilités de blocs. Une méthode optimale pour ce faire a ensuite été trouvée par David Huffman en 1951. Les premières implémentations ont généralement étaient faites dans le matériel, avec des choix spécifiques code.

Au milieu des années 1970, l'idée de mise à jour dynamique pour le codage de Huffman a émergé, sur la base des données réelles rencontrées. Et à la fin des années 1970, avec le stockage en ligne de fichiers texte, les programmes de compression de logiciels ont commencés à être développés, presque tous basés sur le codage adaptatif de Huffman.

En 1977, Abraham Lempel et Jacob Ziv ont suggéré l'idée du codage basés sur des pointeurs. Au milieu des années 1980, suite aux travaux de Terry Welch, l'algorithme LZW est rapidement devenu la méthode de choix pour la plupart des systèmes de compression à usage général. Il a été utilisé dans des programmes tels que PKZIP, ainsi que dans des dispositifs matériels tels que des modems. À la fin des années 1980, les images numériques sont devenues plus fréquentes. Au début des années 1990, les méthodes de compression avec perte ont également commencé à être largement utilisées.

III)-Définition et intérêt de la compression

III-1)-Définition [1]

La compression consiste à réduire la taille physique de blocs d'informations. Un compresseur utilise un algorithme qui sert à optimiser les données en utilisant des considérations propres au type de données à compresser; un décompresseur est donc nécessaire pour reconstruire les données originelles grâce à l'algorithme inverse de celui utilisé pour la compression.

La méthode de compression dépend intrinsèquement du type de données à compresser on ne compressera pas de la même façon une image qu'un fichier audio...

III-2)-Principe de la compression

Les capacités de stockage mais également les bandes passantes (Internet) sont aujourd'hui trop peu développées pour contenir des fichiers de plus en plus lourds. Il s'avère donc nécessaire de réduire la taille de ces fichiers afin de les stocker ou bien encore afin de les transférer plus facilement.

III-3)-Intérêt de la compression

De nos jours, la puissance des processeurs augmente plus vite que les capacités de stockage, et énormément plus vite que la bande passante des réseaux, car cela demande d'énormes changements dans les infrastructures de télécommunication.
Ainsi, pour pallier ce manque, il est courant de réduire la taille des données en exploitant la puissance des processeurs plutôt qu'en augmentant les capacités de stockage et de transmission des données.

Le coût et les limites technologiques nécessitent d'utiliser la compression de données pour le stockage d'importants volumes d'information.

Pour une durée donnée, la compression permet de faire circuler plus d'informations, le débit est donc plus grand.

IV)-La théorie de l'information et ces éléments [2]

La théorie de l'information est due à Shannon (vers 1948), avec bien sur l'influence des grands théoriciens de l'informatique (Turing, Von Neumann, Wiener).

Cette théorie s'intéresse, à la mesure de la quantité de l'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 de l'image en signaux électromagnétique.

IV-1)-Quantité d'information [2]

On considère une source s qui produit des mots aléatoires indépendants les uns des autres et qui peuvent prendre K valeurs possibles m_k avec pour chacun d'entre eux une probabilité p(m_k) (k = 1…K) On définit la *quantité d'information* liée au mot m par:

$$\mathbf{I(m)= -Log_2[p(m)]}$$

IV-2)-Entropie d'un texte [2]

En générale, l'entropie mesure le degré de désordre dans un système, en théorie de l'information il indique le poids d'informations 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 :

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

Remarque:

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

IV-3)-La redondance [2]

La source X est dite sans redondance si tous les éléments de X apparaissent avec la même probabilité.

(p(x1) = p(x2) = p(x3) =… p(xn)).

La redondance d'une source caractérise la différence qu'il existe entre la quantité d'information que transporté cette source et la quantité d'information que cette source transporterait si tous ces symboles étaient équiprobables et indépendants.

On à:

$$\mathbf{R = 1-(H(x)/Hmax\ (x))}$$

V)-Le codage et le code

V-1)-Le codage [3]

Le codage permet d'établir une correspondance qui permet sans ambigüité de passer d'une représentation (dite externe) d'une information à une autre représentation (dite interne : sous forme binaire) de la même information, suivant un nombre de règle précise.

EXEMPLE :

Le nombre 15 : donc 15 est la représentation externe du nombre quinze (15).

La représentation interne de 15 sera une suite de 0 et 1 (1111)

V-1-1)-Classification de codage

Il existe trois (03) types de codage :

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

Codage correcteur d'erreurs : est utilisé pour la correction d'erreurs de transmission.

Codage de la source : est utilisé pour la compression des données.

V-2)-Le Code [3]

Un sous-ensemble non-vide X de A* est un code quand tout mot de A* s'écrit d'au plus une façon comme concaténation de mots de X, X est un code si et seulement si quand On a :

x1 · · · xn = y1 · · · ym

OÚ : les xi et les yi sont des mots de X, alors n = m et pour tout i ∈ {1, . . . n}, xi = yi.

On n'impose pas que tout mot soit représentable comme concaténation de mots de X, mais que s'il l'est alors il l'est d'une unique façon.

V-2-1)-Types de code [2]

V-2-1-1)-Code à longueur variable

Le code à longueur variable est un code qui associe les symboles de la source à un nombre variable de bits.

Les codes à longueur variable peuvent permettre à la source d'être compressée et décompressée avec une erreur nulle: il s'agit d'une compression sans perte. L'opération inverse du codage est alors possible pour chaque symbole. À partir d'une bonne stratégie de codage, une source peut être compressée arbitrairement proche de son entropie, ce qui permet par exemple d'associer des mots longs aux symboles sources les moins fréquents. C'est la différence avec les méthodes de codage à longueur fixe.

Parmi les codes à longueur variables possibles pour effectuer un codage de source, nous allons considérer les codes préfixes.

Un arbre k-aire est un arbre tel que chaque nœud a au plus k fils. Il est dit k-aire complet si chaque nœud a exactement k-fils et si tout nœud a au moins un descendant qui est une feuille.

La suite génératrice des feuilles d'un arbre est la suite $(\boldsymbol{S_n})\mathbf{n\geq 0}$, où sn est le nombre de feuilles de hauteur n, la hauteur étant la distance à la racine.

Lorsque l'arbre est fini, tous les sn sont nuls à partir d'un certain rang. On représente parfois la suite $(\boldsymbol{S_n})\mathbf{n\geq 0}$ par la série formelle $\mathbf{s(z)} = \sum_{n\geq 0} s_n Z^n$.

Le théorème de Kraft-McMillan donne une condition nécessaire et suffisante à l'existence d'un arbre k-aire dont la série génératrice des feuilles est donnée. On dit qu'une suite s = (sn)n≥0 satisfait l'inégalité de Kraft pour l'entier k ssi $\sum_{n\geq 0} s_n k^{-n} \leq 1$,
Ou encore si la série s(z) associée vérifie S (1/K) ≤ 1.

V-2-1-2)-Code à longueur fixe

Le code à longueur fixe est un code qui associe les symboles de la source à un nombre fixe de bits.

Si une source a pour cardinal n, il est possible de la coder avec un code de longueur fixe m tel que $log_2 n \leq m \leq 1 + log_2 n$

L'efficacité E d'un code de longueur m est égale à $H(A)/\ m$. Comme $H(A)\ \leq\ log2\ n$,

On a E ≤ 1 et E= 1 seulement si :

$H(A) = log_2 n$ C'est-à-dire les lettres de la source sont équiprobables,

m = $log_2\ n$ c'est-à-dire le cardinal de la source est une puissance de 2.

V-3)-Déchiffrabilité d'un code

Un code C sur un vocabulaire V est dit uniquement déchiffrable (on dit parfois non ambigu) si et seulement si, pour tout $X = X_1..X_n \in V^*$.il existe au plus une séquence C $c_1 \ldots c_n$ telle que $c_1 \ldots..c_n = X_1 \ldots \ldots.X_n$

Propriété :

Un code C sur un vocabulaire V est uniquement déchiffrable si et seulement si pour tout séquence c=$c_1 \ldots \ldots\ c_n$ et d=$d_1 \ldots \ldots\ d_n$ de C^*

$$\mathbf{C{=}d \rightarrow (n = m\ et\ 1 \leq i \leq n, c_i = d_i)}$$

VI)-La compression et la décompression

La compression consiste à coder les données d'un fichier d'une manière plus compacte que l'original afin de diminuer son encombrement. Tandis que la décompression consiste à rétablir le fichier compressé dans son état initial.

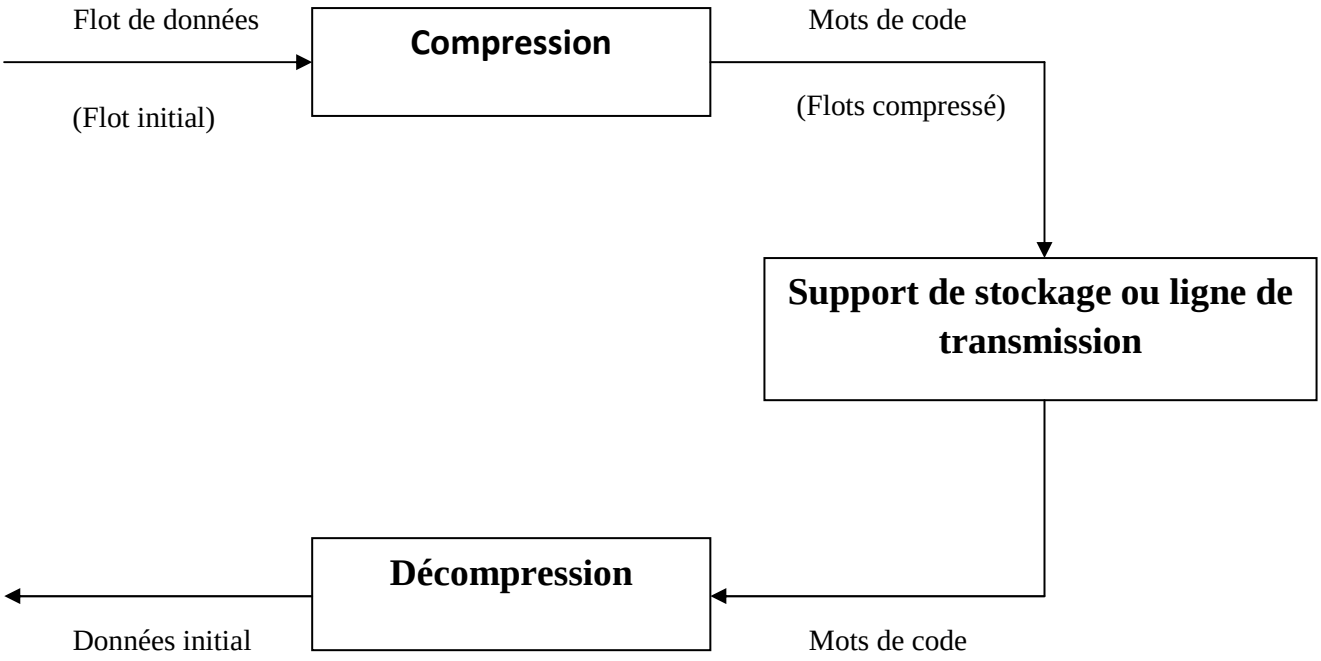


Figure I : Schéma générale de la compression/décompression

VI-1)-La différence entre le codage et la compression

Le codage est l'envoi des données de l'émetteur vers le récepteur par le canal sous forme de signaux (des codes), tandis que la compression de données consiste à réduire la taille de la donnée qu'on veut envoyer à l'émetteur pour réduire le temps de transmission.

VI-2)-Les types de compression :

Il existe deux(2) types de compression

VI-2-1)-Compression sans perte [4]

Ce type de compression s'applique à des données dont on ne doit pas modifier le contenu et qu'il est essentiel de garder dans leur intégralité. C'est la méthode de compression pour les fichiers de données type texte ou binaire. Il y a autant d'information après la décompression qu'avant la compression.

C'est également le cas pour les programmes et autres fichiers exécutables qui ont besoin de chacune des informations qu'ils contiennent pour fonctionner.

VI-2-2)-Compression avec perte [5]

Elle admet la perte des données considérées redondantes ou inutiles en échange d'un meilleur taux de compression. Un fichier compressé puis décompressé selon cette méthode sera différent de l'original. Les images, les sons et la vidéo utilisent la compression avec perte.

VI-3)-Les types de données à compresser [6]

Il existe plusieurs types de données à compresser

VI-3-1)-Le texte

De manière générale, toute information structurée en ASCII ou composée de caractères alphanumérique.

Dans un traitement de texte et en PAO (Publication Assisté par Ordinateur), partie principale d'un document, par opposition aux titres, tableau, figure, notes de bas de page et autres éléments.

VI-3-2)-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 don un signal que l'on peut représenter sous forme d'une courbe mathématique indiquant l'intensité en fonction du temps qui s'appelle un signale analogique ce signale doit encore être numérisé pour pouvoir être exploiter par l'ordinateur pour cela, il est échantillonné, c'est-à-dire découpé dans le temps , par une carte son.

VI-3-3)-L'image

C'est une représentation plane d'un objet tridimensionnelle, perçu par œil humain, ou plus généralement un capteur dont le fonctionnement est semblable (exemple d'objet : une scène, un portrait, une échographie, une observation astronomique … etc). 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 est une fonction d'intensité lumineuse ou de couleur.

VI-3-3-1)-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 points par ligne, soit de nombre de lignes, soit le produit des deux.

VI-3-3-2)-La numérisation

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

OÚ :

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

VI-3-3-3)-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).

VI-3-3-4)-Mode couleur

RGB (Red-Green-Blue) ce triplet permet de définir une couleur : chaque couleur peut prendre une valeur de 0 a 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) représente le blanc . C'est ce qu'on appelle la synthèse additive (rouge+vert+rouge) à laquelle échelle additive s'imagine facilement en ajoutant des couleurs à la couleur noire. Donc plus on ajoute de couleurs (R, V, B), plus on se rapproche du blanc. Elle est utilisée pour des écrans.

Elle s'oppose à l'échelle soustractive, ou 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 couleurs, la photographie . on trouve également d'autres modèles, comme le YUV(lumière et chrominance) ,utilisé pour les téléviseurs (et le format JPEG), et HSV(ou HSB) :HUE, saturation,brightness.

VII)- Classification des algorithmes de compression [7]

VII-1)-Compression symétrique / asymétrique

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.

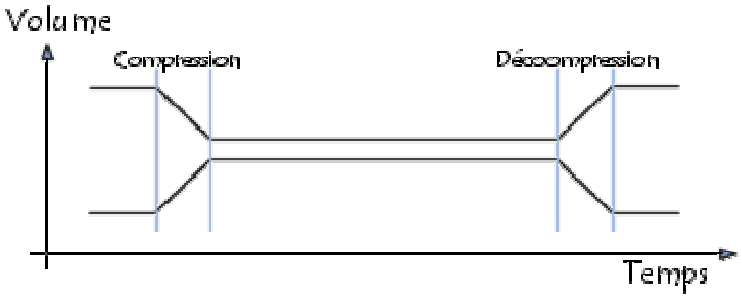


Figure II : Schéma de la compression symétrique

La compression **asymétrique** demande plus de travail pour l'une des deux opérations, on recherche souvent des algorithmes pour lesquels la compression est plus lente que la décompression. Des algorithmes plus rapides en compression qu'en décompression peuvent être nécessaires lorsque l'on archive des données auxquelles on n'accède peu souvent (pour des raisons de sécurité par exemple).

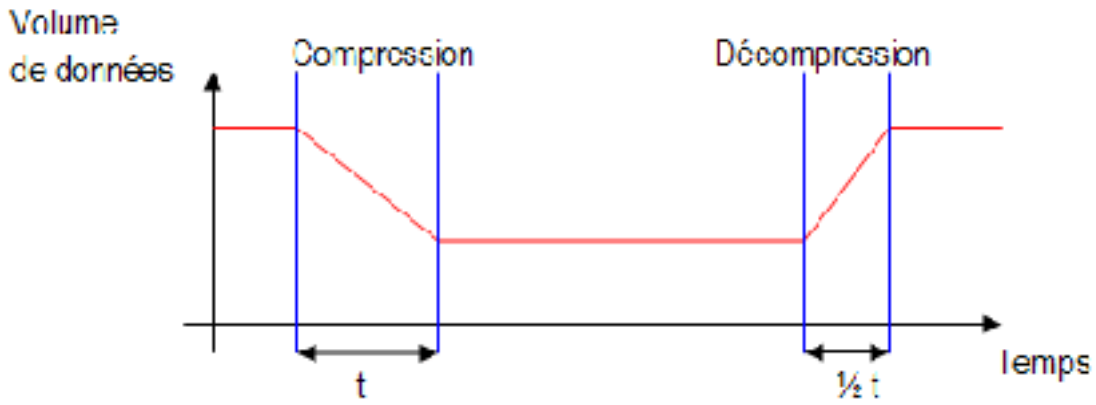


Figure III : Schéma de la compression asymétrique

VII-2)-Compression physique / logique

On considère généralement la compression comme un algorithme capable de comprimer des 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 se base sur la façon dont les données sont compressées ou plus précisément comment est-ce que les données sont réarrangées.

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.

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.

Exemple :

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.

VII-3)-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, 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 les seconds, 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.

VII-4)-Encodage adaptif, semi adaptif et non adaptif

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

Un encodeur **adaptif** s'adapte aux données qu'il va devoir compresser, il ne part pas avec un dictionnaire déjà préparé pour un type de données.

Enfin un encodeur **semi-adaptif** construira celui-ci en fonction des données à compresser : il construit le dictionnaire en parcourant le fichier, puis compresse ce dernier.

VII-5)-La compression sans / avec perte

VII-5-1)-Compression sans perte [8]

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.

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

VII-5-1-1)- Algorithmes a base de la modélisation statique

Calculent les probabilités pour aider à la compression, typiquement, en tentera d'estimer le plus précisément possible la fonction de distribution de la prochaine donnée, de

façon à pouvoir généré un code efficace pour cette prochaine donnée en théorie ils peuvent mener à une compression optimale.

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.

VII-5-1-2) algorithmes à base de transformation

Appliqueront plutôt une transformation sur les données qui mettra en évidence les répétitions, de façon à pouvoir mieux les exploités pour la compression, ces algorithmes sont souvent suivis d'une autre étape de compression comme méthode à base de dictionnaire ou statique.

VII-5-1-3)- Algorithmes à base de dictionnaire

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

Le but est de :

- Réduire le nombre de bits utilisés pour le codage des mots fréquents.
- Augmenter ce nombre pour des mots plus rares.

Remarque :

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.

VII-5-2)-Compression avec perte [8]

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. Ce type de compression engendre une dégradation indiscernable à l'œil (ou à l'oreille) ou suffisamment faible, en contrepartie d'un taux de compression très élevé.

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, ...).

VIII)- Performance de compression [9]

VIII-1)-Taux de compression

Le taux de compression τ est relié au rapport entre la taille b du fichier comprimé B et la taille a du fichier initial A. Le taux de compression est généralement exprimé en pourcentage. Un taux de 50 % signifie que la taille b du fichier comprimé B est la moitié de a. La formule pour calculer ce taux est :

$$\tau = 1 - (b/a)$$

Exemple : a=550 Mo, b=250 Mo

$$\tau = 1 - (250/550) = 54\%$$

L'algorithme utilisé pour transformer A en B est destiné à obtenir un résultat B de taille inférieure à A. Il peut paradoxalement produire parfois un résultat de taille supérieure : dans le cas des compressions sans pertes, il existe *toujours* des données incompressibles, pour lesquelles le flux compressé est de taille supérieure ou égale au flux d'origine.

VIII-2)-Le gain de compression

Le gain de compression est l'espace dont on a bénéficié après l'opération de compression. Tant que le gain est important, la compression est efficace.

Il est donné par la formule suivante :

Gain de compression = 100 * taux de compression

VIII-3)-Vitesse de compression

C'est le temps nécessaire à la compression d'un flux de données, dont la taille est précise.

IX)- Conclusion

La compression de données est appelée à prendre un rôle encore plus important en raison du développement des réseaux et du multimédia. Son importance est surtout due au décalage qui existe entre les possibilités matérielles des dispositifs que nous utilisons (débits sur Internet et sur les divers câbles, capacités des mémoires de masse, …) et les besoins qu'expriment les utilisateurs (visiophonie, vidéo plein écran, transfert de quantités d'information toujours plus importantes dans des délais toujours plus brefs). Quand ce décalage n'existe pas, ce qui est rare, la compression permet de toute façon de faire des économies.

Les méthodes déjà utilisées couramment sont efficaces et sophistiquées (Huffman, LZW, JPEG) et utilisent des théories assez complexes. Les méthodes émergentes sont prometteuses (fractales, ondelettes) mais nous sommes loin d'avoir épuisé toutes les pistes de recherche. Les méthodes du futur sauront sans doute s'adapter à la nature des données à compresser et utiliseront l'intelligence artificielle.

Chapitre II
Compression par arbre

I)-Introduction

Dans ce chapitre on s'intéressera aux méthodes de compression/décompression de données et les algorithmes correspondants basés sur les arbres, ce sont des méthodes de codage de la source d'information.

La compression ne doit pas conduire à des résultats erronés et doit être réalisée avec les taux les plus élevés possible et on un temps court.

II)-Définitions

II-1)-Un alphabet

Un alphabet est un ensemble de symboles destiné à représenter plus ou moins précisément les phonèmes d'une langue. Chacun de ces symboles, est aussi appelé lettre.

II-2)-Un mot

Un mot est une suite de sons ou de caractères graphiques formant une unité sémantique et pouvant être distingués par un séparateur.

II-3)- Arbre binaire [11]

Par définition un arbre binaire est un arbre avec une racine, et où chacun des nœuds possède :

- soit aucun successeur,
- soit un successeur, à gauche ou à droite,
- soit deux successeurs (un à gauche, et un à droite)

Voici un exemple d'arbre binaire, dessiné avec sa racine en haut et ses branches en descente, à l'inverse d'un arbre dans la nature :

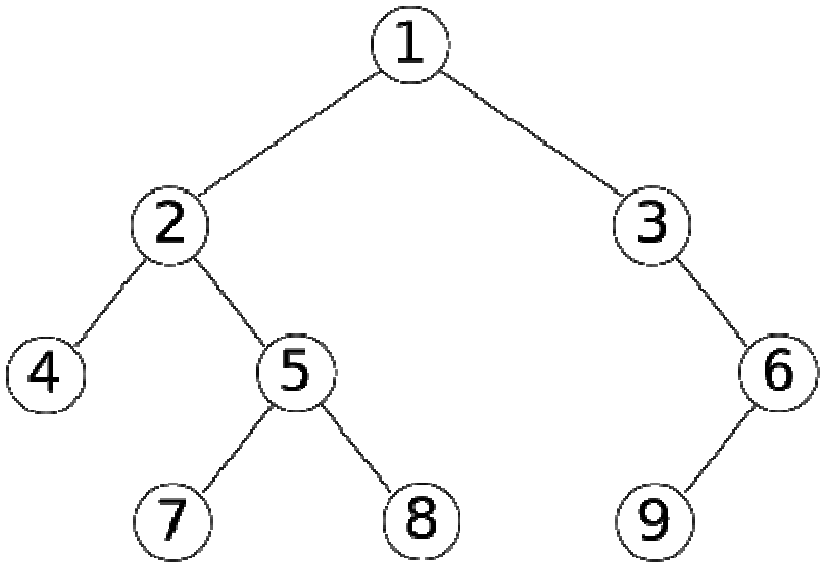


Figure I : Exemple d'un arbre binaire

II-4)-Le code préfixe

Un code préfixe (ou code instantané) est un code ayant la particularité de ne posséder aucun mot ayant pour préfixe un autre mot. Autrement dit, aucun mot fini d'un code préfixe ne peut se prolonger pour donner un autre mot.

C'est une propriété souvent recherchée pour les codes à longueur variable, afin de pouvoir les décoder lorsque plusieurs mots sont concaténés les uns aux autres sans qu'il soit nécessaire d'utiliser des séparateurs (les séparateurs rendent préfixes des codes non préfixes). Ce sont des codes non ambigus.

Remarque :

Les codes à taille fixe sont tous des codes préfixes.

II-5)-Le codage [12]

Le codage est l'opération consistant à transformer une suite de bits A en une suite de bits B plus courte.

Exemple de codage

Coder le texte suivant :

S : « abcbacaad »

- Calcul de l'alphabet : A={a,b,c,d}.
- Construction de l'arbre binaire et calcul des codes pour les caractères de l'alphabet.

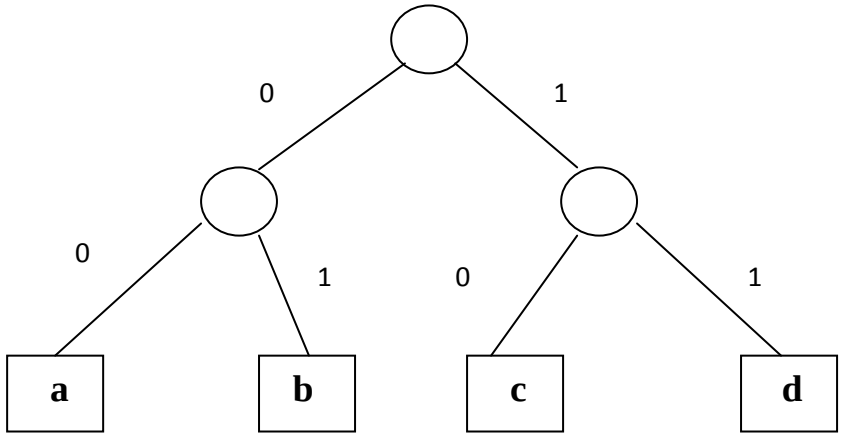


Figure II : Exemple de code

Table des codes :

Caractère	code
a	00
b	01
c	10
d	11

Génération du code associé à notre texte en remplaçant les caractères par leurs codes :

« abcbacaad » ⟶ {00-01-10-01-00-10-00-00-11} (taille=18)

II-6)-Le décodage [12][13]

Le décodage est l'opération inverse du codage qui consiste à restituer le mot d'origine à partir du mot codé en utilisant l'opération inversé.

II-7)-Le codage préfixe optimal [12]

Le codage préfixe optimal consiste à construire un arbre, dans lequel les caractères les plus fréquents posséderont les codes les plus courts.

Exemple: Reprenons l'exemple précédent

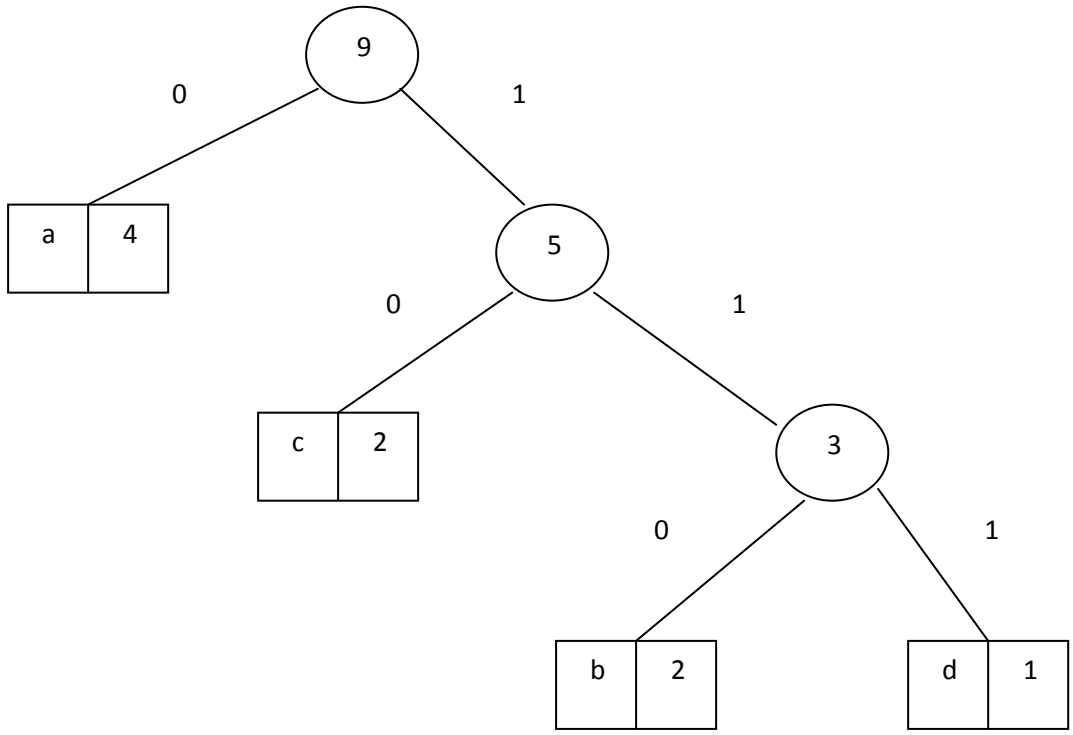


Figure III : Arbre de codage préfixe optimale

Table des codes :

Caractère	code
a	0
b	110
c	10
d	111

Génération du code associé à notre texte en remplaçant les caractères par leurs codes :

« abcbacaad » ⟶ {0-110-10-110-0-10-0-0-111} (taille=17)

II-8)-Codage entropique [12]

Le codage entropique (ou codage statistique à longueur variable) est une méthode de codage de source sans pertes, dont le but est de transformer la représentation d'une source de données pour sa compression ou sa transmission sur un canal de communication.

II-9)-Le calcul

L'entropie est la quantité d'informations contenues dans un message, et se note de la manière suivante :

Entropie : ***I=log2(z)***

Z : est ici le code décimal du message.

II-10)-Parcours d'un arbre binaire [13]

On se balade autour de l'arbre en suivant les pointillés dans l'ordre des numéros indiqués :

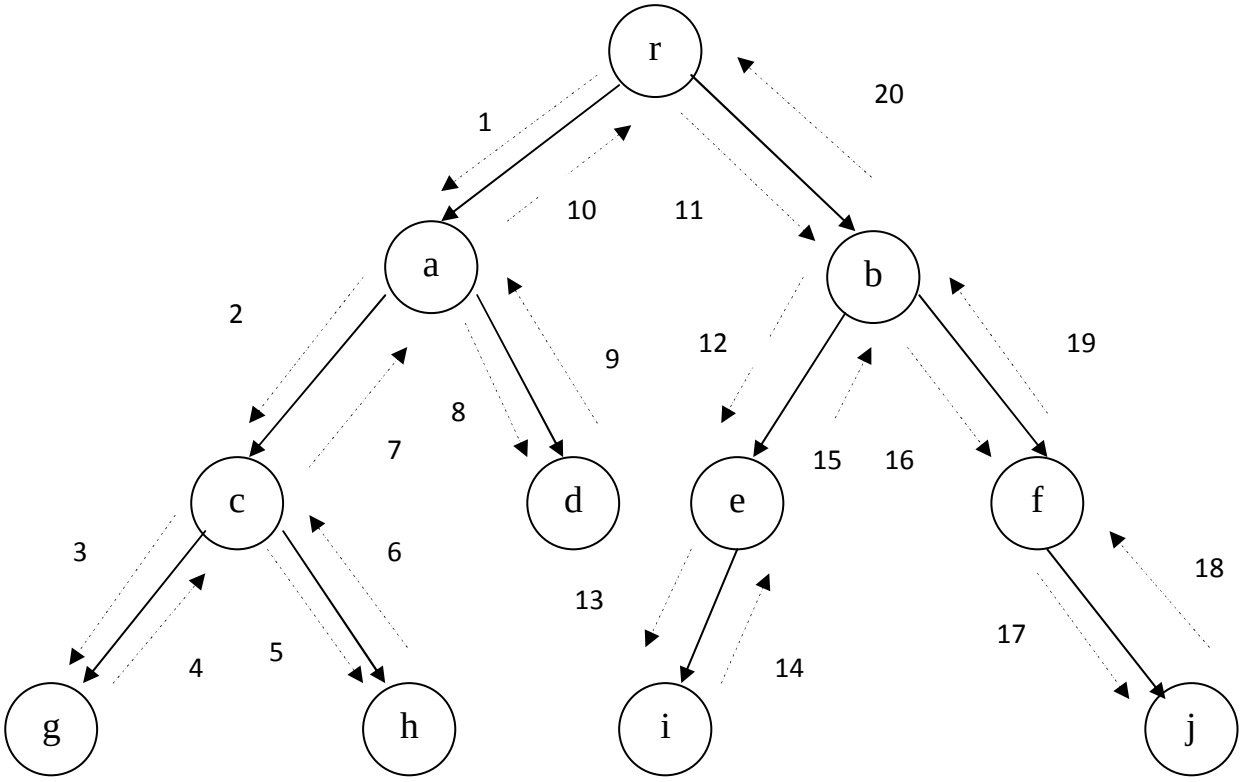


Figure IV : Schéma de parcours d'un arbre binaire

A partir de ce contour, on définit trois parcours des sommets de l'arbre :

1. **l'ordre préfixe** : on liste chaque sommet la première fois qu'on le rencontre dans la balade. Ce qui donne ici **: *r*, *a*, *c*, *g*, h, *d*, *b*, *e*, i, f, j.**

Pseudo code :

```
ParcoursPréfixe (Arbre binaire T de racine r)
        Afficher c l e f [r]
        ParcoursPréfixe (Arbre de racine fils_gauche [r])
        ParcoursPréfixe (Arbre de racine fils_droit[r])
```

2. **l'ordre postfixe** : on liste chaque sommet la dernière fois qu'on le rencontre. Ce qui donne ici : **g, h, c, d, a, i, e, j, f, b, r.**

Pseudo code :

```
ParcoursPostfixe (Arbre binaire T de racine r)
        ParcoursPostfixe (Arbre de racine fils_gauche [r])
        ParcoursPostfixe (Arbre de racine fils_droit[r])
        Afficher c l e f [r]
```

3. **l'ordre infixe** : on liste chaque sommet ayant un fils gauche la seconde fois qu'on le voit et chaque sommet sans fils gauche la première fois qu'on le voit. Ce qui donne ici **: g, c, h, a, d, r, i, e, b, f, j.**

Pseudo code :

```
ParcoursInfixe (Arbre binaire T de racine r)
        ParcoursInfixe (Arbre de racine fils_gauche[r])
        Afficher c l e f [r]
        ParcoursInfixe (Arbre de racine fils_droit[r])
```

III)-Algorithme de HUFFMAN :

III-1)-Définition [14]

Le codage de Huffman est un algorithme de compression des données 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 A. 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%.

III-2)-Principe de compression [14][15]

L'algorithme de Huffman utilise une table contenant les fréquences d'apparition de chaque caractère pour établir d'une manière optimale leur représentation par une chaîne binaire.
On peut décomposer la procédure en plusieurs parties :
– Tout d'abord, la création de la table de fréquence d'apparition des caractères dans le texte initial.
– 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.)
– Enfin coder les symboles en représentation binaire optimale.

- **Table de fréquence d'apparition**

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

fs =nombre d'occurrences de s /nombre de symboles

Ensuite on trie le tableau en fonction de la fréquence d'apparition (de façon croissante) puis suivant le symbole suivant.
Pour chaque symbole on calcule tout d'abord le nombre d'occurrences de chacun puis sa fréquence d'apparition suivant la formule citée précédemment.

- **Construction de l'arbre de Huffman**

L'arbre binaire de Huffman est la structure de données qui va nous permettre d'attribuer à chaque symbole une représentation binaire optimale. A fin de construire l'arbre, on utilise la table de fréquences et on applique l'algorithme suivant :

<u>Algorithme 1</u>: 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 du 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 noeud 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 2 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.

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 a 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 efficace.

III-3)-Exemple de compression :

Soit le texte suivant à compresser en utilisant l'algorithme de huffman

« L'algorithme de huffman sert à compresser des données en utilisant des arbres binaires »

- **Table de fréquence d'apparition**

Nombre de symbole du texte est : 87.

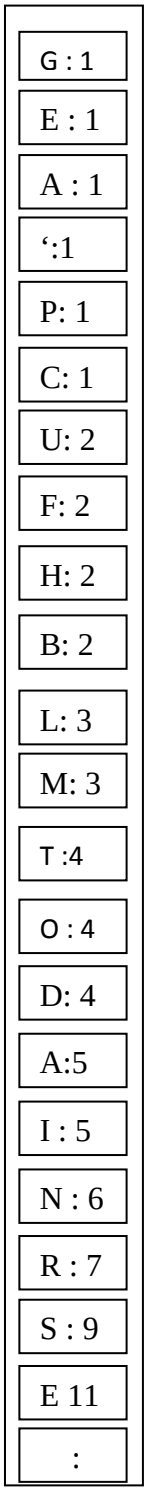
Symbole	Nombre d'occurrences	Fréquence d'apparition
G	1	0, 012
É	1	0.012
Â	1	0.012
‘	1	0.012
P	1	0.012
C	1	0.012
U	2	0.024
F	2	0.024
H	2	0.024
B	2	0.024
L	3	0.037
M	3	0.037
O	3	0.037
A	5	0.048
D	4	0.048
T	4	0.048
I	5	0.061
N	6	0.061
R	7	0.086
S	9	0.096
E	11	0.123
	13	0.160

- **Construction de l'arbre de Huffman**

A laide de l'algorithme de construction de l'arbre de Huffman en construiT l'arbre étape par étape :

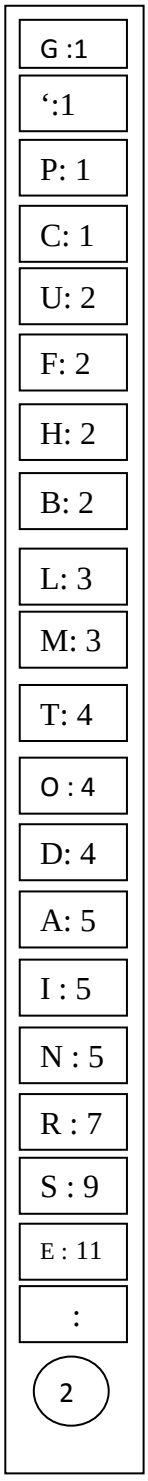
Initialisation de la pile

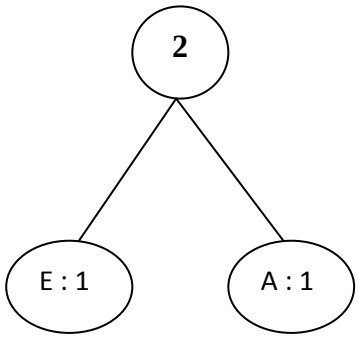
Q



Etape 1

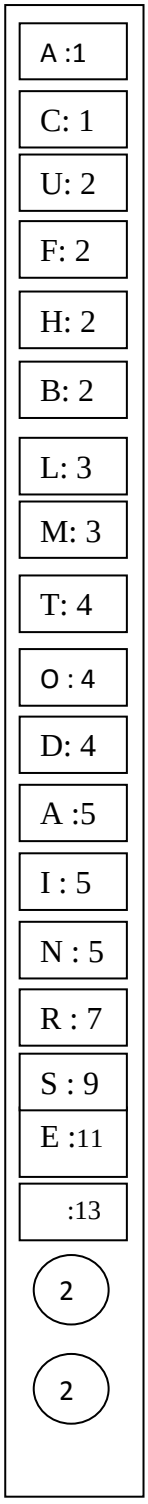
Q

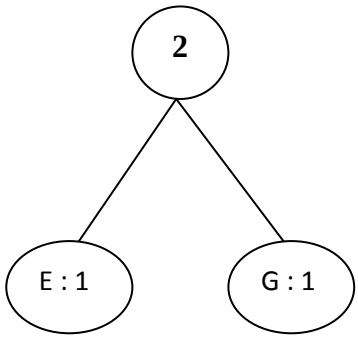


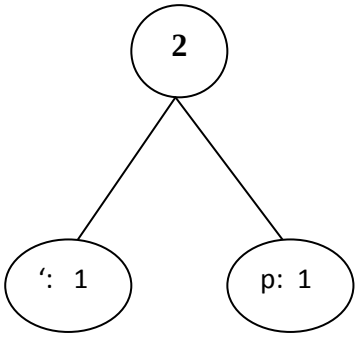


Etape 2

Q







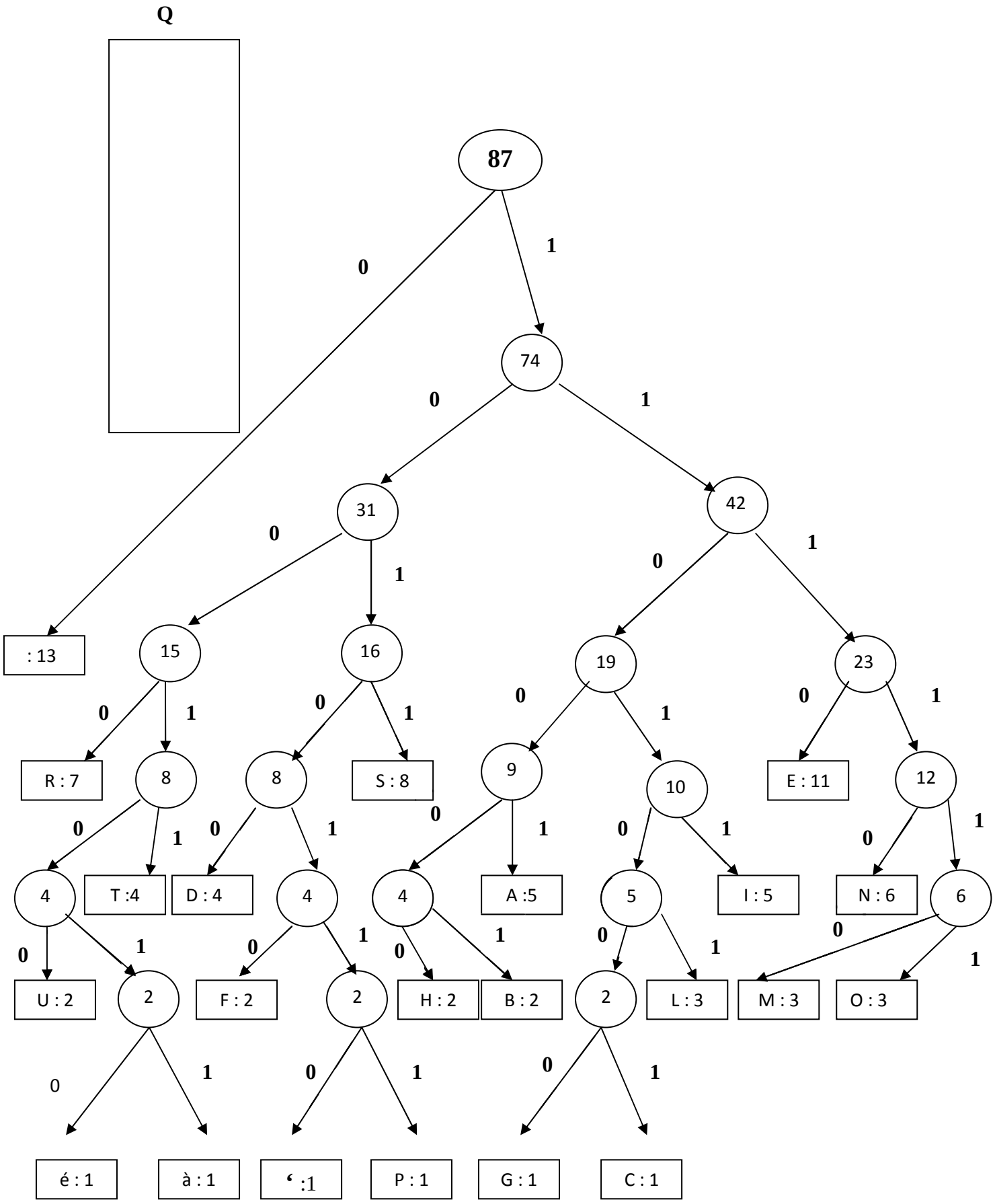


Figure V : Arbre de HUFFMAN

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’.

A partir de l’arbre on obtient la table de correspondance suivante :

Symbole	Représentation binaire
E	1001010
A	1001011
‘	1010110
P	1010111
C	1101001
U	100100
F	101010
H	110000
B	110001
L	110101
M	111110
O	111111
A	11001
D	10100
T	10011
I	11011
N	11110
R	1000
S	1011
E	1110
	0
G	1101000

III-4)-L a décompression

Lors de la réception d’un message, le destinataire récupère dans l’entête l’arbre de Huffman et substitue chaque suite de bits correspondant à un chemin dans l’arborescence de Huffman par le symbole équivalent. On remarque qu’il ne peut pas y’avoir d’ambiguïté sur l’interprétation d’une séquence de bits à cause de la structure intrinsèque de l’arborescence.

III-5)-Performances

L’intérêt de la compression est de pouvoir réduire au maximum la taille des informations originales, comme nous l’avions indiqué en première partie.

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

$$\boldsymbol{P = 1 - \frac{taille\ compressé}{taille\ originale}}.$$

Par ailleurs, on peut aussi utiliser l'entropie permet de connaître le nombre minimum de bits nécessaires au codage d'un fichier. On rappelle que la formule de l'entropie est la suivante :

$$\boldsymbol{E = \sum_{N}^{1} (P_k * log_2 (P_k))}$$

Où : $\boldsymbol{P_k}$ 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 tout d'abord puis nous généraliserons.

III-6)-Performances générales

Les performances sont dépendantes de la fréquence d'apparition des symboles dans le message original, par conséquent elles dépendent également du type de fichier que nous désirons compresser. En effet, les fichiers de type exécutable par exemple contiennent, de façon générale, moins de redondances qu'un fichier de type BMP (bitmap pour les images). De cette remarque, on en déduit que le taux de compression sera en moyenne meilleur pour les fichiers où les redondances sont fortes.
Des études ont déjà été réalisées afin de déterminer les taux de compression, voici un aperçu de ce que ces dernières nous montrent :

Type de fichier	Taux de compression
Texte	49.5%
Bitmap (image)	50%
Wave(image)	50%
Exécutable	20%

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.

III-7)-Performances sur l'exemple

Taille originale

Nombre de caractères de la source, chaque caractère dans le code ASCII est sur 8 bits.

Taille originale = 87*8 = 696 bits

Taille compressée

La somme des mots écrits en binaire.

Taille compressée = 391

Taux de compression

$$\mathbf{P = 1-\frac{391}{696} =0,56}$$

Le taux de compression dans cet exemple est de 56%.

IV)-Codage de Shannon Fano [15] [16]

Le codage de Shannon-Fano est un algorithme de compression de données sans perte élaboré par Robert Fano à partir d'une idée de Claude Shannon.

Il s'agit d'un codage entropique produisant un code préfixe très similaire au code de Huffman, bien que pas toujours optimal, contrairement à ce dernier.

IV-1)-Algorithme de Shannon Fano

L'algorithme comporte les étapes suivantes :

1) Les probabilités d'apparition de chaque symbole sont placées dans un tableau trié par ordre décroissant de probabilités.

2) Le tableau est coupé en deux groupes de symboles S0 et S1 dont la somme des probabilités de chaque groupe avoisine 0.5.

3) Le groupe S0 est codé par un "0" et S1 par un "1".

4) Si un groupe Si n'a qu'un seul élément, c'est une feuille terminale, sinon la procédure reprend récursivement à l'étape 2 sur le groupe Si.

IV-2)-Exemple d'application de l'algorithme Shannon-Fano

Pour illustrer cet algorithme, nous allons coder la phase suivante : "**le codage est indispensable**"

Cette phrase est une source de 27 symboles.

Les symboles contenus dans cette phrase sont :

$A=\{E\ ,\ S\ ,ESPACE,\ A\ ,D\ ,\ I\ ,\ N\ ,\ L\ ,\ B\ ,G\ ,\ P\ ,T\ ,O\ ,C\}$

Cet alphabet a N=14 symboles :

Construction du tableau

Symboles	Nombre de fois	Probabilités
E	5	5/27
S	3	3/27
ESPACE	3	3/27
A	2	2/27
D	2	2/27
I	2	2/27
N	2	2/27
L	2	2/27
B	1	1/27
G	1	1/27
P	1	1/27
T	1	1/27
O	1	1/27
C	1	1/27

Application successive des étapes 2,3 et 4.

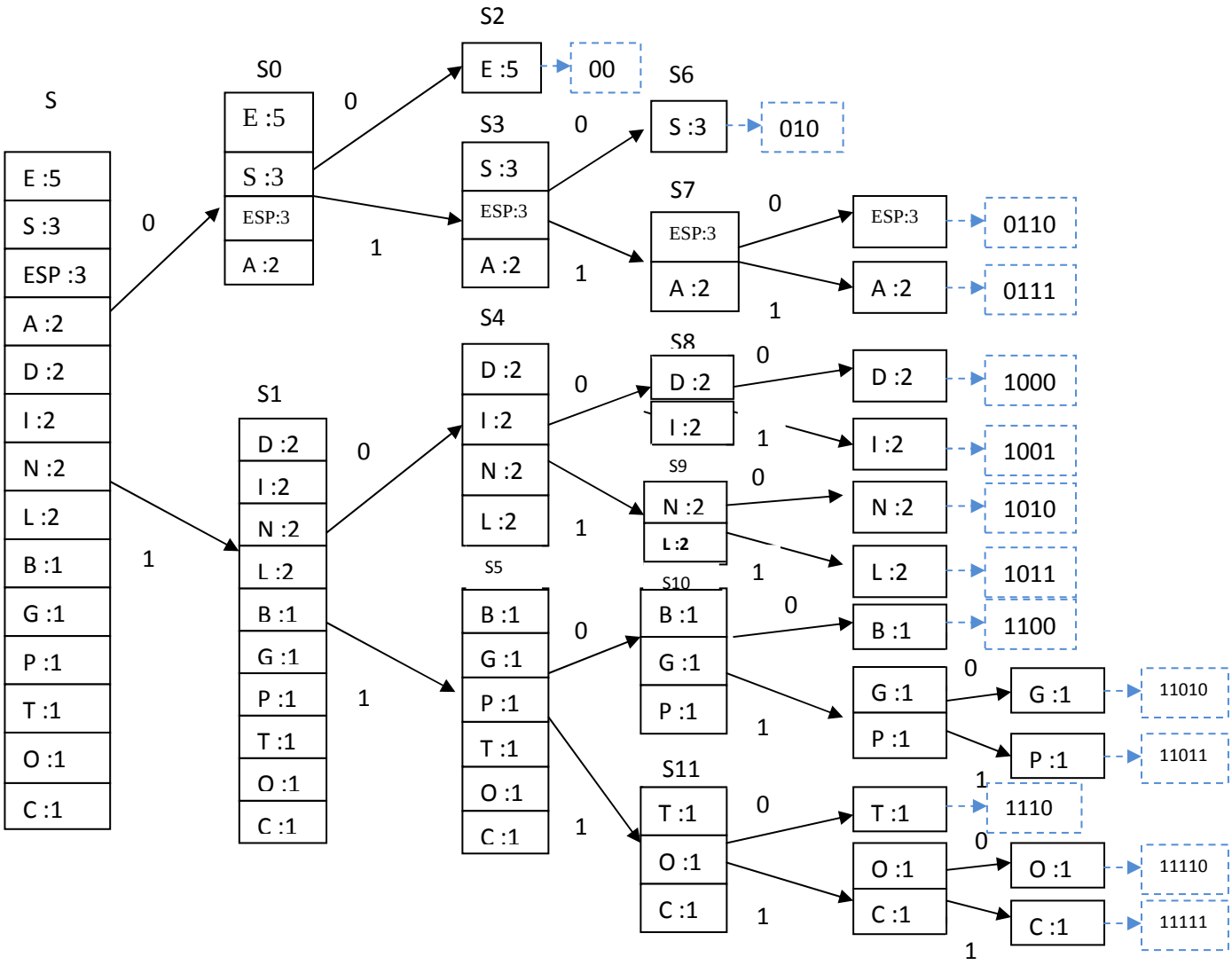


Figure VI : Arbre de Shannon-Fano

V)-Conclusion

Dans ce chapitre nous avons présenté les différentes méthodes de compression basées sur les arbres, qui utilisent le codage préfixe. Parmi elles on trouve les algorithmes de Huffman et de Shannon-Fano, la mise en œuvre de ces deux méthodes est différente, car le codage de Shannon-Fano est récursif, tandis que l'algorithme de Huffman ne l'est pas, il consiste à ajouter des éléments à une table jusqu'à arriver à la fin du texte.

Dans le chapitre qui suit, on présentera la méthode proposée pour la compression de données texte qui est basée sur l'algorithme de Huffman, dans le quel on utilisera un traitement mot par mot, fusionné avec le codage fixe.

Chapitre III

Analyse et conception

I)-Introduction

Après avoir définit, dans les chapitres précédents les différents concepts nécessaires à l'accomplissement de notre travail, nous passons maintenant à la partie analyse et conception.

Dans tout projet informatique la conception est importante et doit être traitée avec précision et en détail, précédée d'une analyse profonde et réfléchie, car elle est le reflet du futur système avant même sa concrétisation. Le modèle orientée objet s'est avérée une approche d'analyse et de conception très puissant et se trouve de plus en plus utilisée. Dans ce chapitre, notre objectif est de modéliser une application de compression de données en orienté objet à l'aide du langage UML (UnifiedModelingLanguage) qui permet de bien représenter les aspects statiques et dynamiques de notre projet par la série des diagrammes qu'il offre.

II)-Présentation de la méthode proposée

Cette méthode est classé parmi les méthodes de compression sans perte, elle est basée sur la création d'un arbre des mots, après avoir lu le fichier à compressé mot par mot, ensuite la compression se fait par la transformation de cet arbre en passant par plusieurs étapes pour enfin le stocker en mémoire.

II-1)-Description de l'application

Notre travail consiste à mettre au point une application de compression de données en implémentant trois (3) méthodes à savoir (codage fixe, codage fixe + 1 bloc HUFFMAN, codage fixe + 3 blocs HUFFMAN). Ensuite étudier leurs performances sur des fichiers de type texte.

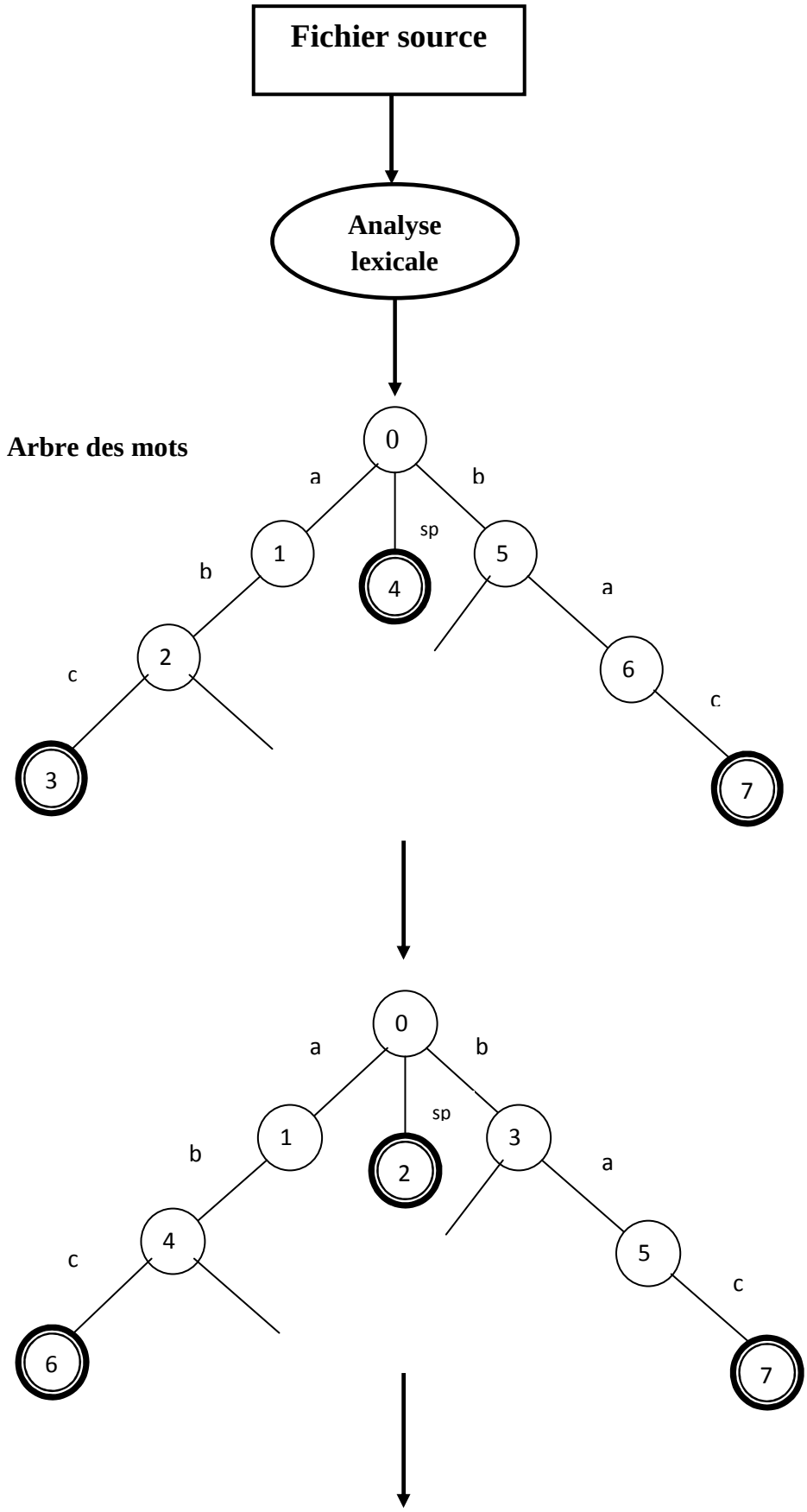
II-2)-Principe de la méthode proposée

II-2-1)-Principe de la compression

La compression est la phase qui consiste à réduire la taille du fichier source.

Cette phase est réalisée comme suit :

1) Lecture du fichier et extraction des mots contenus dans ce dernier.
2) Construction de l'arbre des mots (contient tous les mots du texte).
3) Ordonner l'arbre des mots.
4) Création de la matrice des fréquences (contient tous les mots avec le nombre d'apparition de chacun d'entre eux).
5) Choix de la méthode (selon le nombre de mots).
6) Génération du code de chaque état final.
7) Compression du texte.
8) Sauvegarde du résultat.


Fichier source
Analyse lexicale
Arbre des mots
0
a
b
1
sp
5
b
4
a
2
6
c
c
3
7
0
a
b
1
sp
3
b
2
a
4
5
c
c
6
7

Matrice des fréquences

Etat final	Nombre d'apparitions
2	2
6	2
7	1
…….	…….

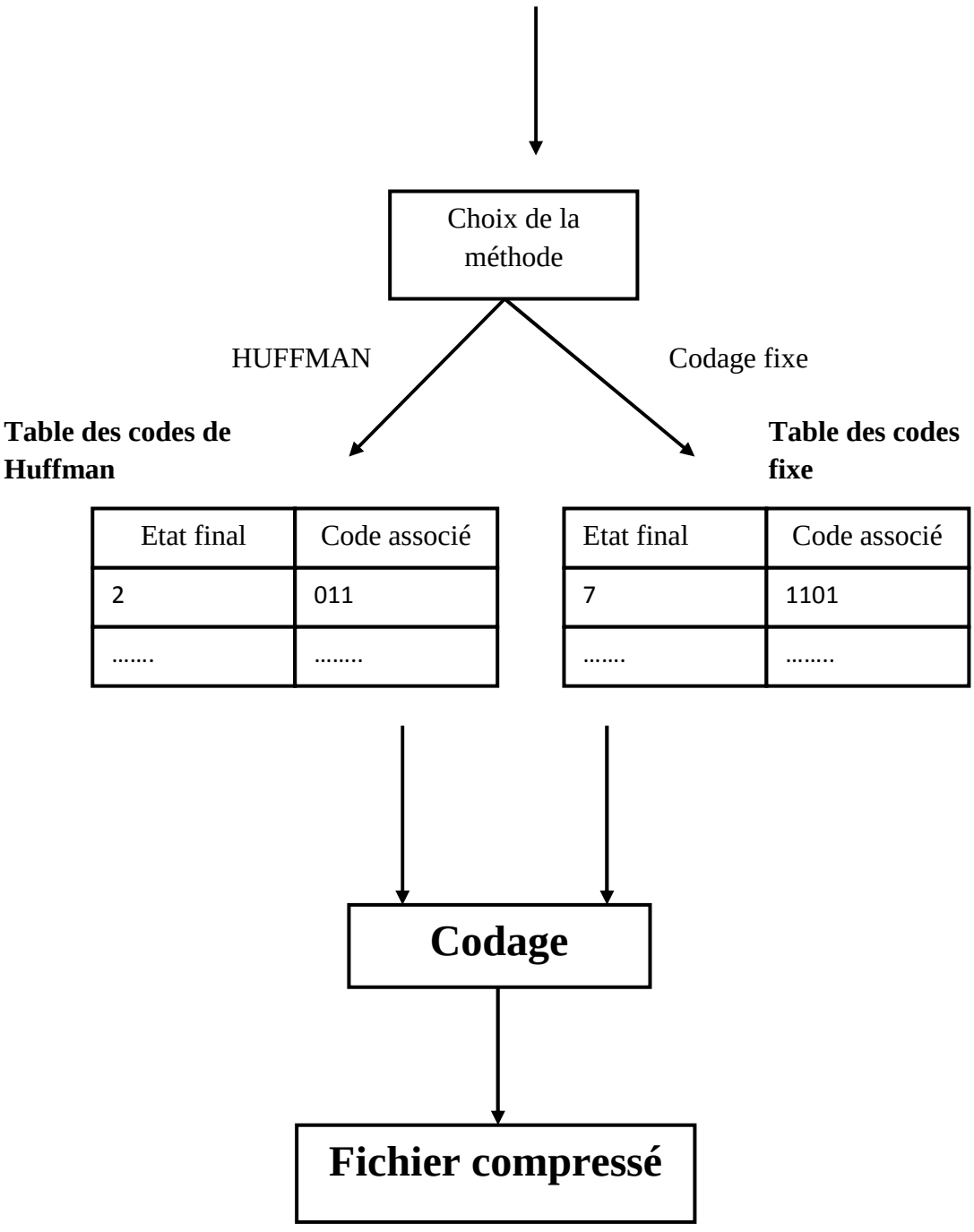


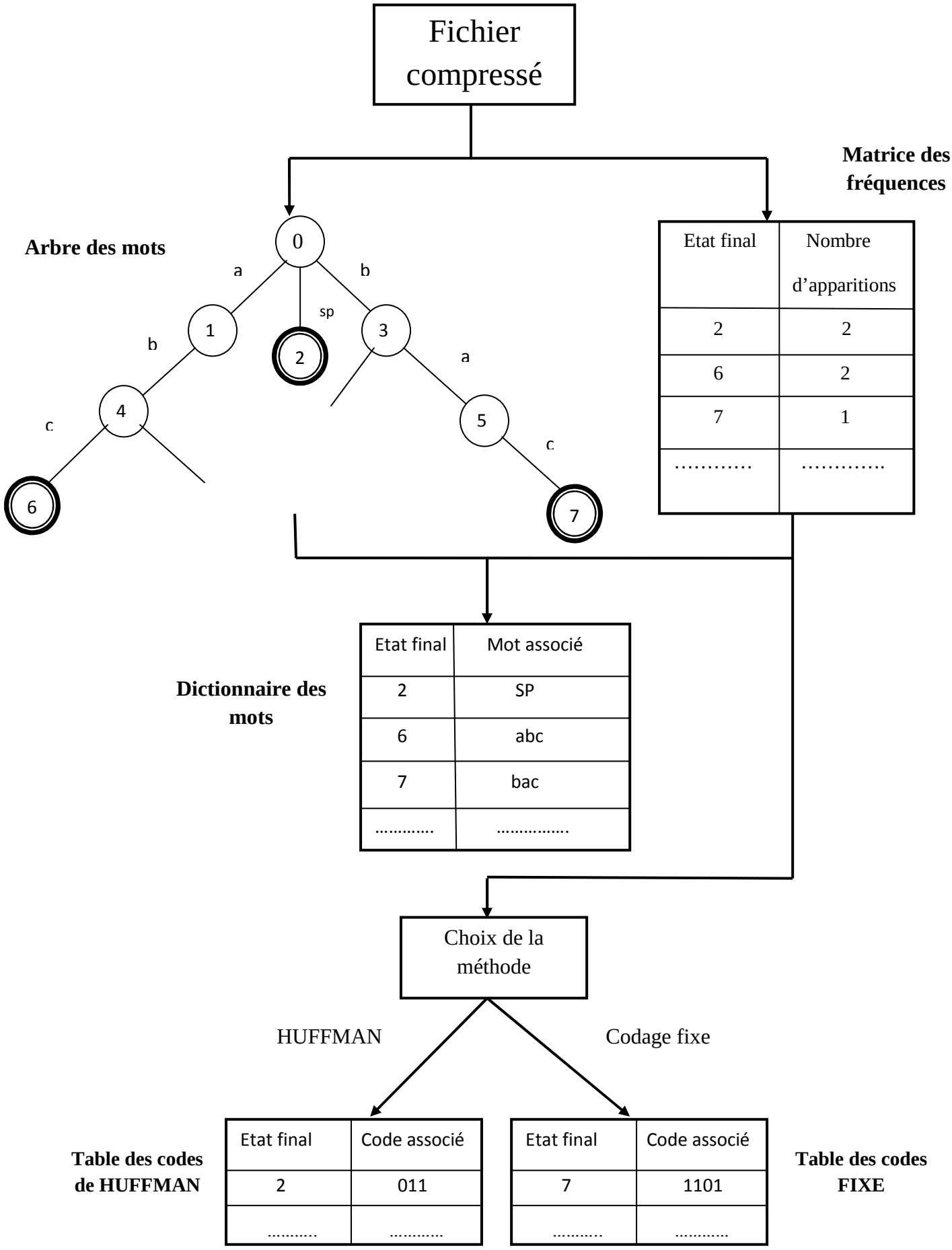
Figure II : Schéma général de la compression

II-2-2)-Principe de la décompression

La décompression est la phase qui consiste à restaurer le fichier source à partir du fichier compressé.

Cette phase est réalisée comme suit :

1) Lecture du fichier et récupération de l'arbre des mots ainsi que la matrice des fréquences.
2) Construction du dictionnaire des mots (contient tous les états finaux associés à leurs mots).
3) Choix de la méthode (doit être la même que celle choisie pour la compression).
4) Génération du code de chaque état final.
5) Décompression du texte.
6) Sauvegarde du résultat.


Fichier compressé
Matrice des fréquences
Arbre des mots
0
a
b
sp
1
2
3
b
a
4
5
c
c
6
7
Etat final
Nombre d'apparitions
2
2
6
2
7
1
Dictionnaire des mots
Etat final
Mot associé
2
SP
6
abc
7
bac
Choix de la méthode
HUFFMAN
Codage fixe
Table des codes de HUFFMAN
Etat final
Code associé
2
011
Etat final
Code associé
7
1101
Table des codes FIXE

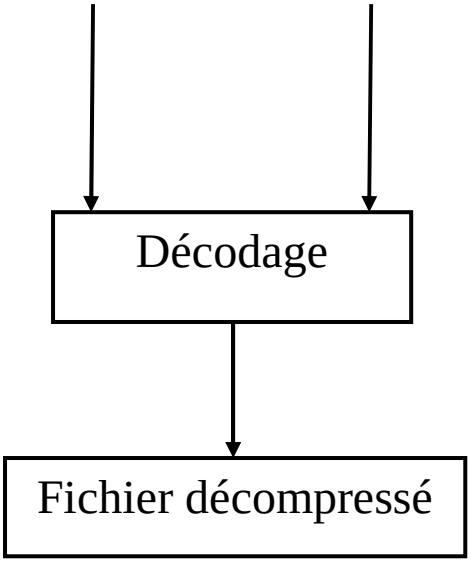


Figure I : Schéma général de la décompression

III)-CONCEPTION DE L'APPLICATION

III-1)-Démarche choisie pour la conception

UML permet de représenter des modèles, mais il ne définit pas de processus d'élaboration de modèles. Pour ce fait nous avons choisi la méthode RUP (Rational Unified Process).

Processus unifié (PU): est une méthode de prise en charge du cycle de vie d'un logiciel et donc du développement pour les logiciels orientés objets.

Rational unified process (RUP) : est l'une des plus célèbres implémentations de la méthode PU permettant de donner un cadre au développement logiciel.

La méthode RUP est :

Itératif et incrémental :

- Chaque itération prend en compte un certain nombre de cas d'utilisation.
- Les risques majeurs sont traités en priorité.
- Chaque itération donne lieu à un incrément et produit une nouvelle version exécutable.

Piloté par les cas d'utilisation

- La principale qualité d'un logiciel est son utilité.
- Adéquation du service rendu par le logiciel avec les besoins des utilisateurs.
- Le développement d'un logiciel doit être centré sur l'utilisateur
- Les cas d'utilisation permettent d'exprimer ces besoins.
- Détection et description des besoins fonctionnels.
- Organisation des besoins fonctionnels.

Centré sur l'architecture

- Modélisation de différentes perspectives indépendantes et complémentaires.

La démarche de modélisation choisie pour concevoir notre application peut être représentée graphiquement comme suite :

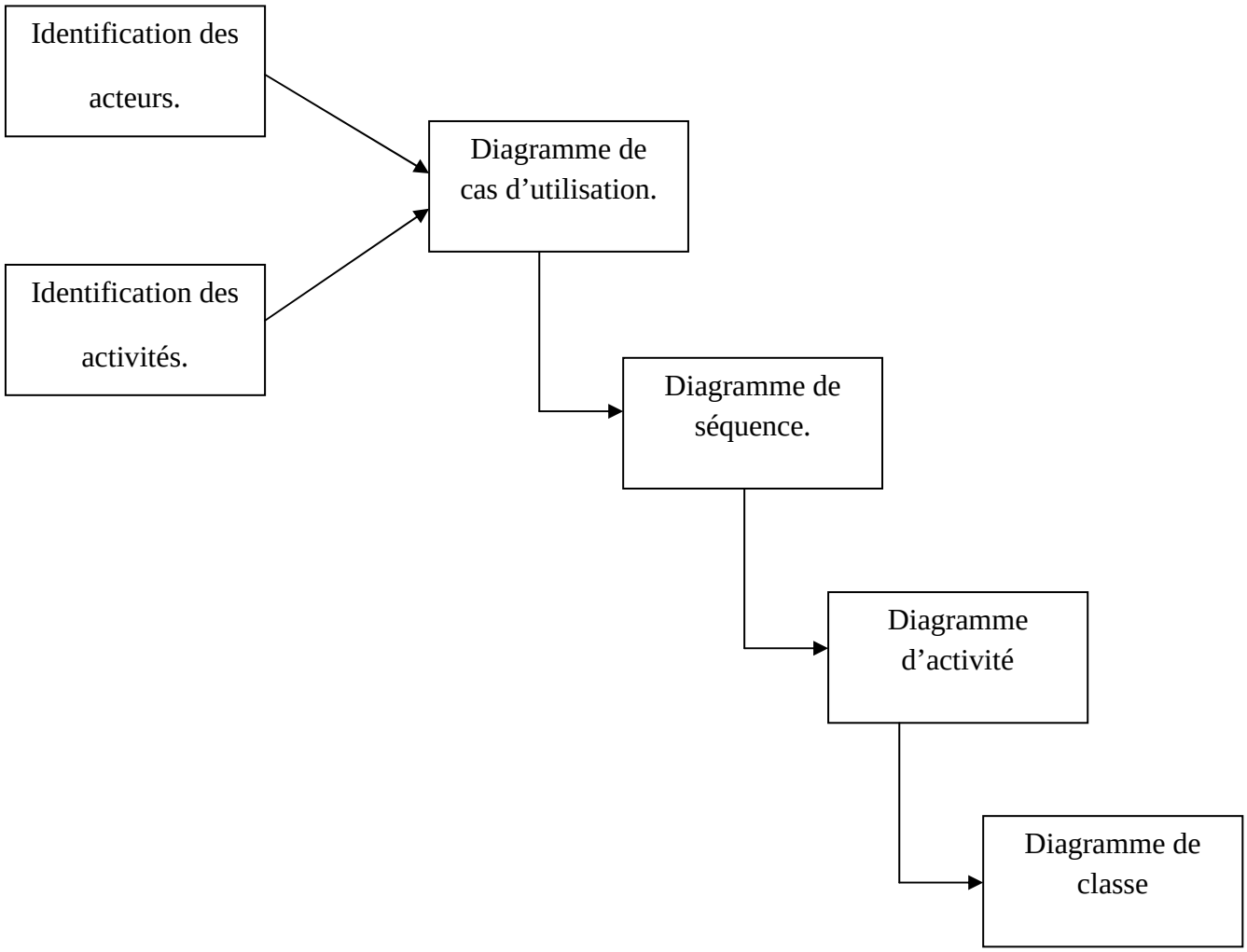


Figure III : Modélisation de l'application

III-1-1)-Identification des acteurs

Les acteurs sont les utilisateurs de l'application.

III-1-2)-Identification des activités

- Compression des fichiers texte.
- Décompression des fichiers texte.
- Affichage des performances (temps, taux).

III-1-3)-Cas utilisation

Diagramme de cas utilisation

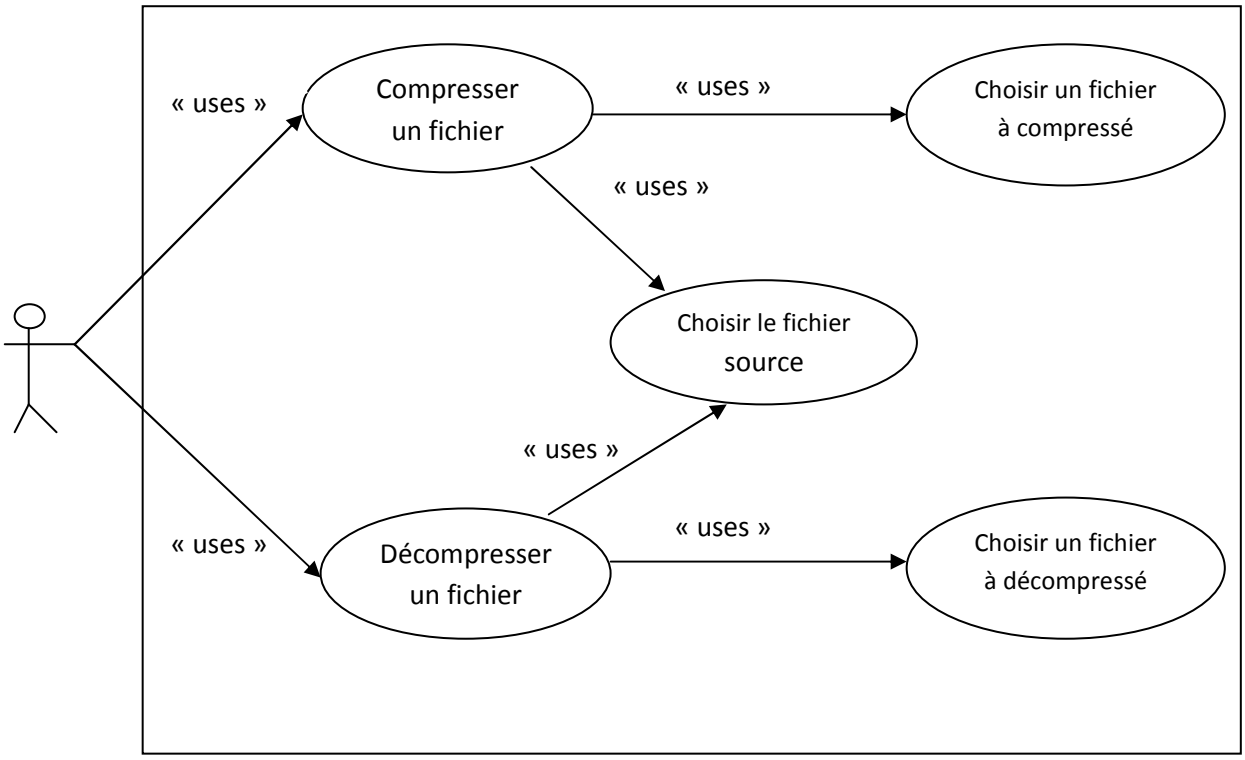


Figure IV: Diagramme de cas utilisation

III-1-3-1)-Description de cas utilisation

Apres avoir identifié les différents cas d'utilisation, nous allons maintenant dresser une description textuelle pour chaque cas d'utilisation en mettant en évidence les acteurs qui l'active, les scénarios contenus et une description des actions à réaliser.

- **Cas utilisation:** compresser un fichier.

Activé par: utilisateur.
Scénario: compresser.

Description :

1. L'utilisateur choisi le fichier à compresser.
2. L'utilisateur choisi le fichier destination.
3. Le système charge le fichier source.
4. Le système choisi la méthode à utiliser.
5. Le système crée le fichier destination.
6. Le système compresse le fichier source.
7. Affichage des performances (temps, taux).

- **Cas utilisation:** décompresser un fichier.

Activé par : utilisateur.
Scénario : décompresser un fichier.

Description :

1. L'utilisateur choisi le fichier à décompresser.
2. L'utilisateur choisi le fichier destination.
3. Le système charge le fichier à décompresser.
4. Le système choisi la méthode à utiliser.
5. Le système crée le fichier destination.
6. Le système décompresse le fichier.
7. Affichage des performances (temps).

III-1-4)-Diagramme de séquence

Les diagrammes de séquences montrent des interactions entre objets selon un point de vue temporel. Dans la description des cas d'utilisation, nous avons pu identifier les scénarios d'utilisation. Dans ce qui suit, nous allons décrire ces derniers en diagrammes de séquence.

III-1-4-1)-Diagramme de séquence (Compression) :

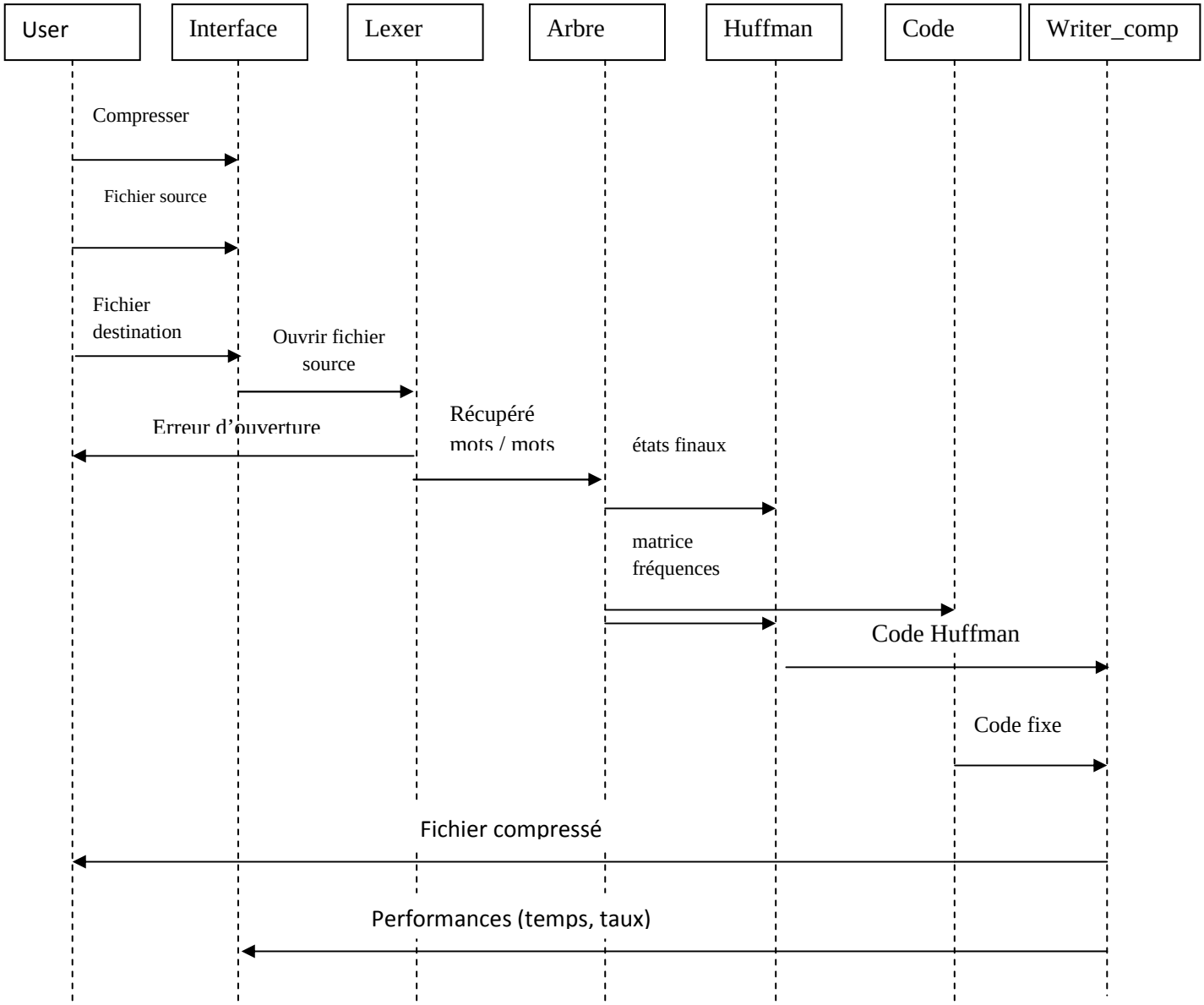


Figure V : Diagramme de séquence pour la compression

III-1-4-2)-Diagramme de séquence (Décompression) :

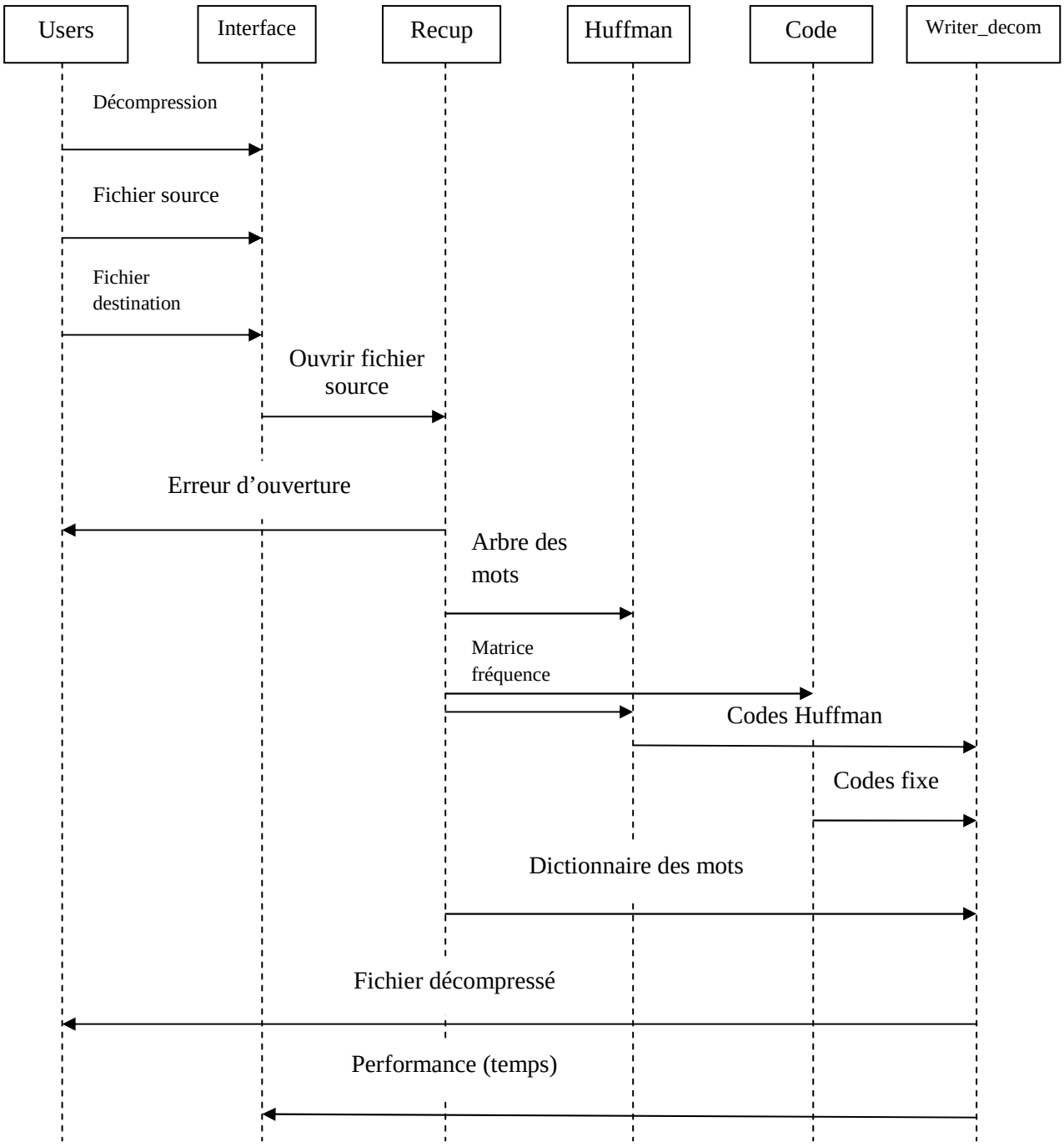
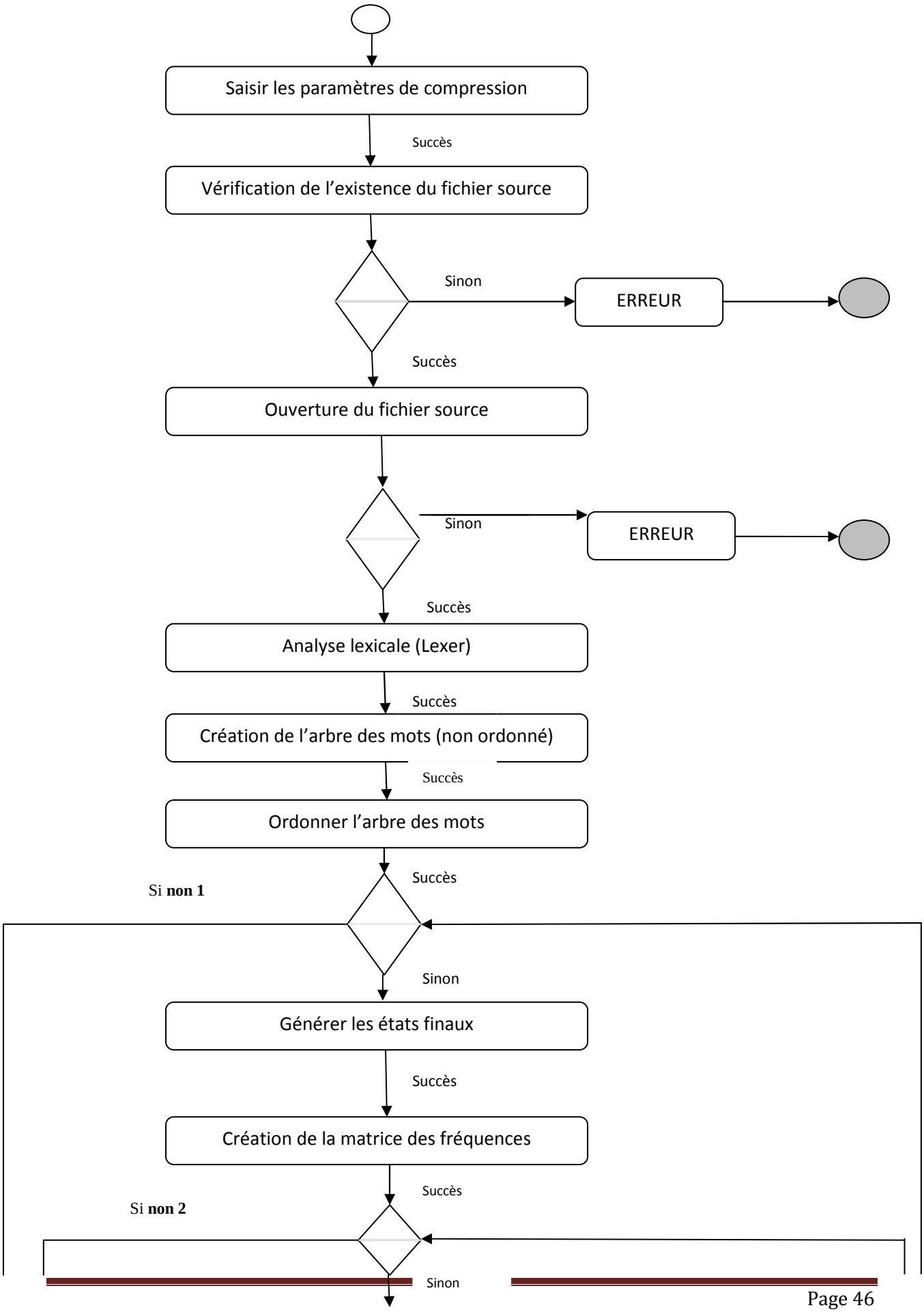
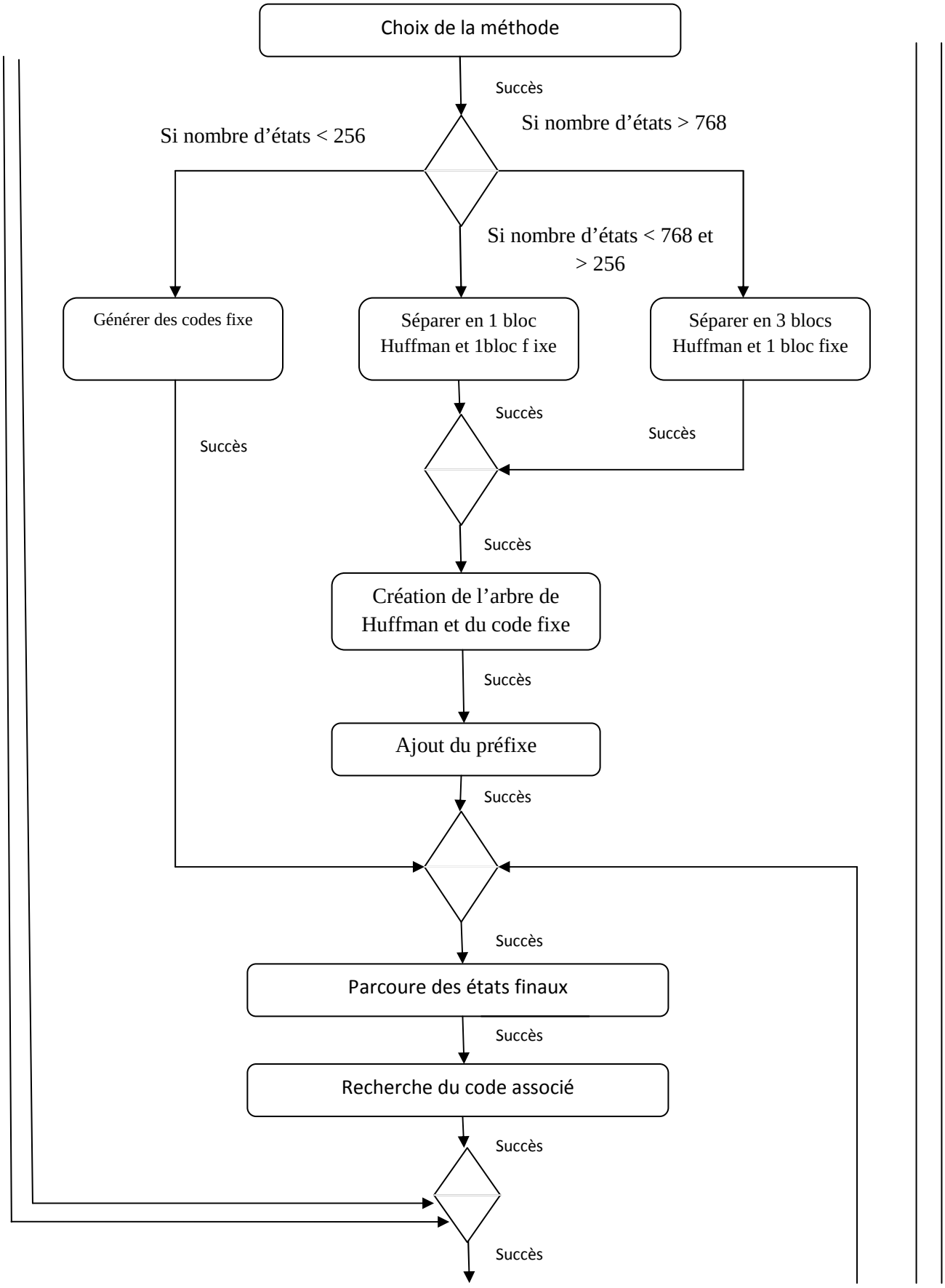


Figure VI : Diagramme de séquence pour la décompression

III-1-5)-Diagramme d'activité

III-1-5-1)-Diagramme d'activité (compressé) :


Saisir les paramètres de compression
Succès
Vérification de l'existence du fichier source
Sinon
ERREUR
Succès
Ouverture du fichier source
Sinon
ERREUR
Succès
Analyse lexicale (Lexer)
Succès
Création de l'arbre des mots (non ordonné)
Succès
Ordonner l'arbre des mots
Succès
Si non 1
Sinon
Générer les états finaux
Succès
Création de la matrice des fréquences
Succès
Si non 2
Sinon


Choix de la méthode
Succès
Si nombre d'états < 256
Si nombre d'états > 768
Si nombre d'états < 768 et > 256
Générer des codes fixe
Séparer en 1 bloc Huffman et 1bloc f ixe
Séparer en 3 blocs Huffman et 1 bloc fixe
Succès
Succès
Succès
Succès
Création de l'arbre de Huffman et du code fixe
Succès
Ajout du préfixe
Succès
Succès
Parcoure des états finaux
Succès
Recherche du code associé
Succès
Succès

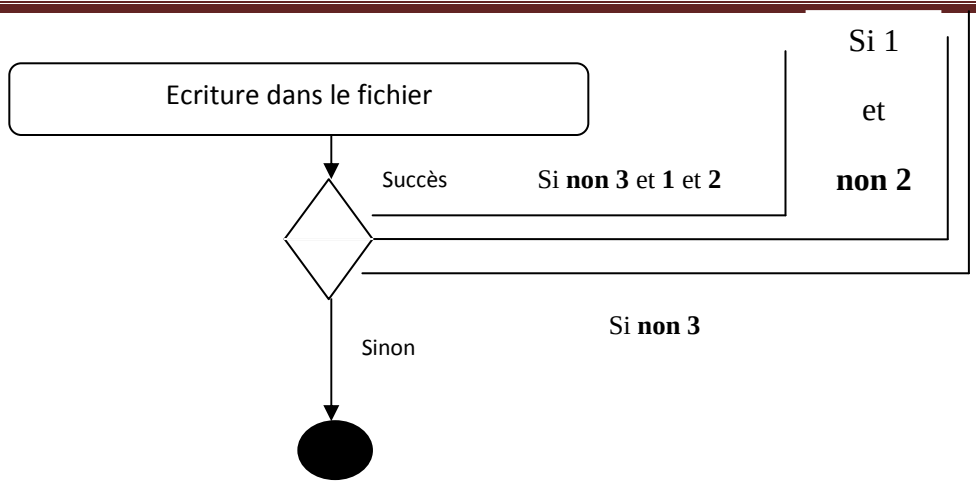


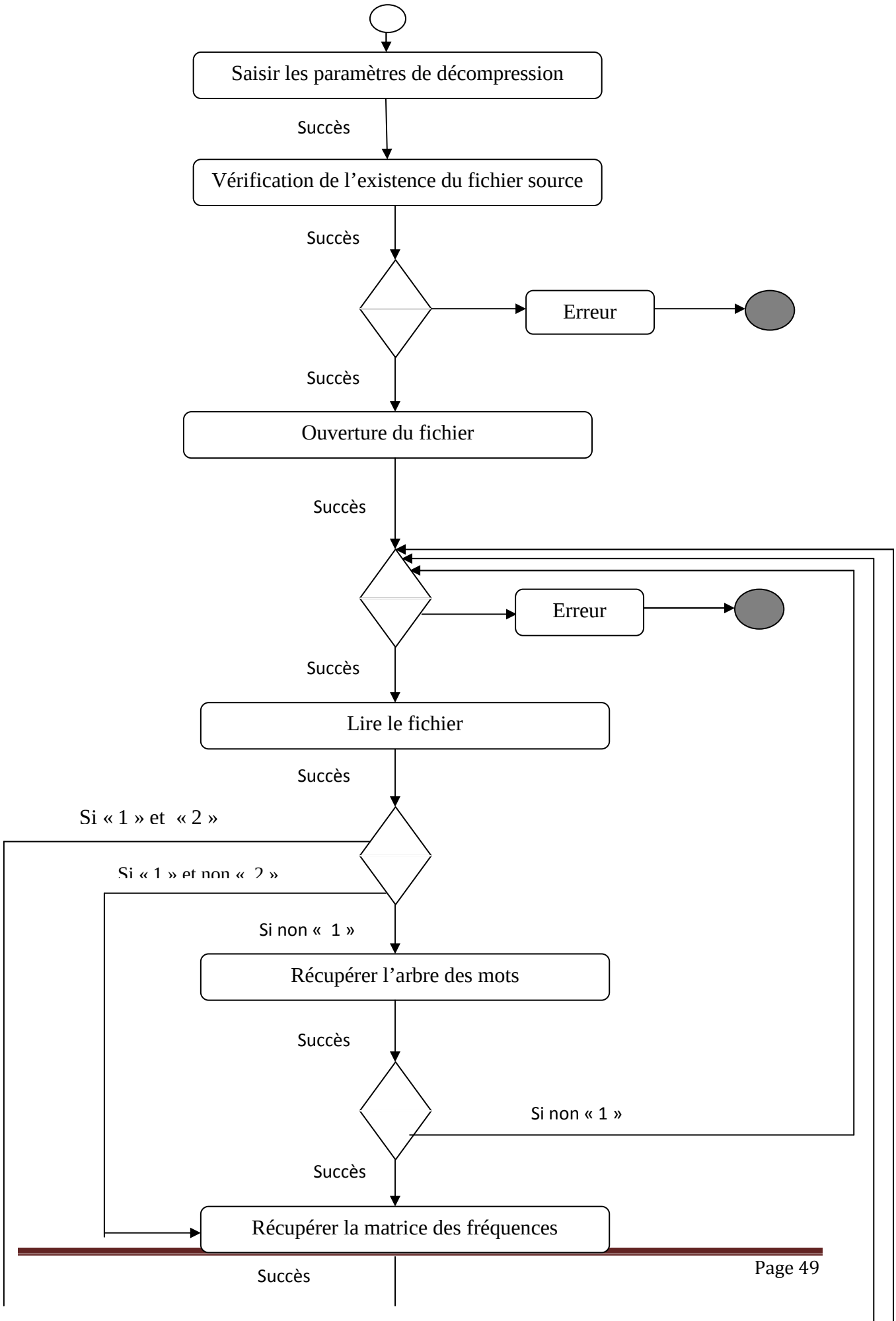
Figure VII : Schéma du diagramme d'activité (compressé)

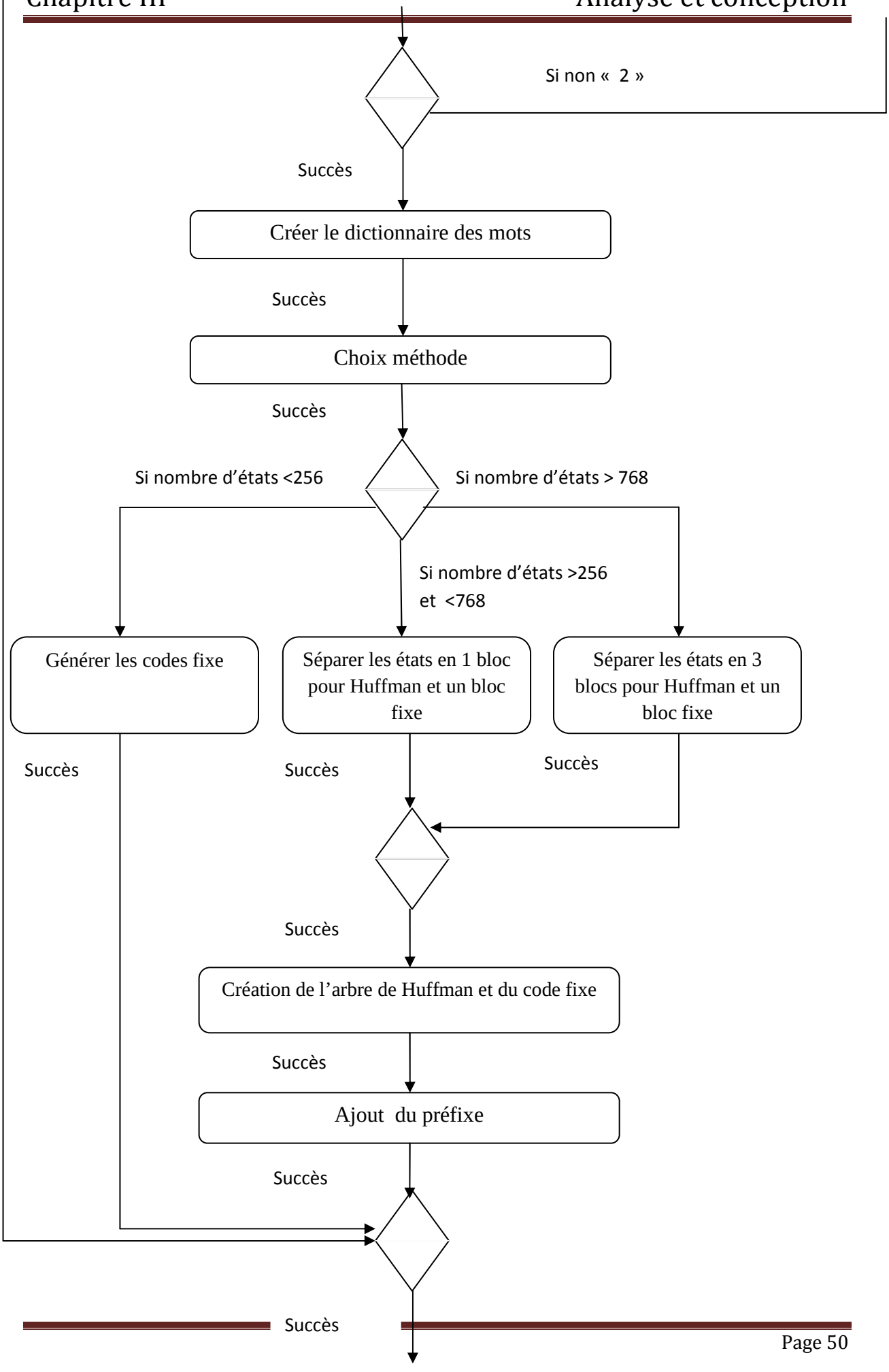
1: Nous avons écrit tout l'arbre.

2 : Nous avons écrit toute la matrice.

3 : Nous avons codifié tous les états.

III-1-5-2)-Diagramme d'activité (décompressé) :


Saisir les paramètres de décompression
Succès
Vérification de l'existence du fichier source
Succès
Erreur
Succès
Ouverture du fichier
Succès
Erreur
Succès
Lire le fichier
Succès
Si « 1 » et « 2 »
Si « 1 » et non « 2 »
Si non « 1 »
Récupérer l'arbre des mots
Succès
Si non « 1 »
Succès
Récupérer la matrice des fréquences
Succès


Si non « 2 »
Succès
Créer le dictionnaire des mots
Succès
Choix méthode
Succès
Si nombre d'états <256
Si nombre d'états > 768
Si nombre d'états >256 et <768
Générer les codes fixe
Séparer les états en 1 bloc pour Huffman et un bloc fixe
Séparer les états en 3 blocs pour Huffman et un bloc fixe
Succès
Succès
Succès
Succès
Création de l'arbre de Huffman et du code fixe
Succès
Ajout du préfixe
Succès
Succès

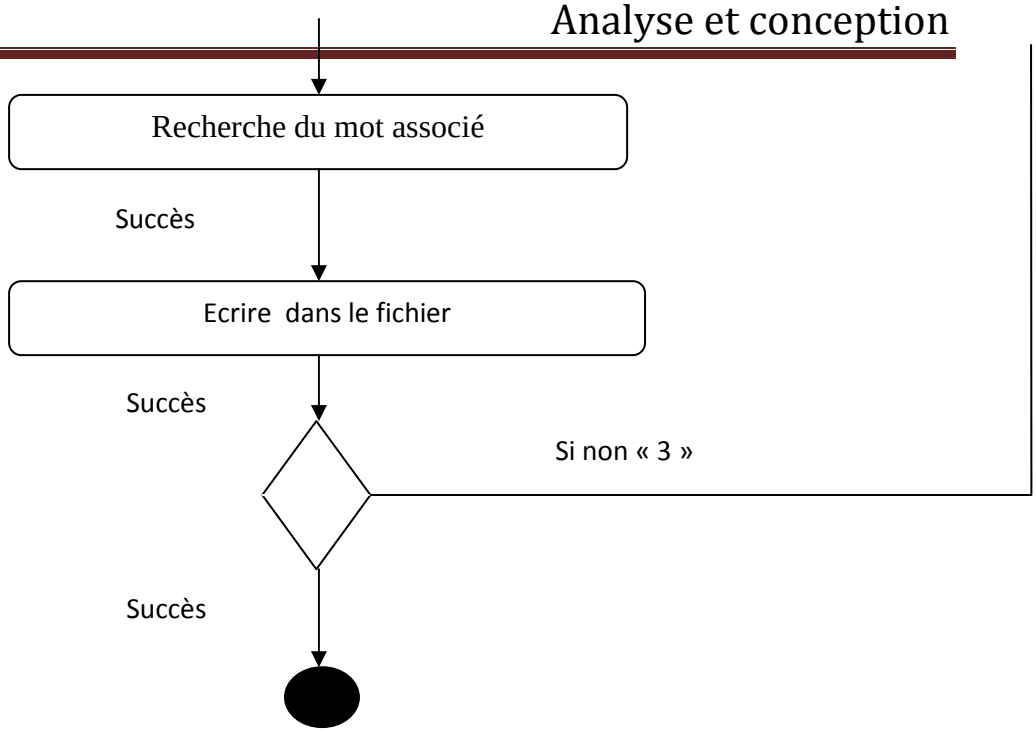


Figure VIII : Schéma du diagramme d'activité (décompressé)

1 : Arbre des mots récupérer.

2 : Matrice des fréquences récupérer.

3 : texte récupérer.

III-1-6)-Diagramme de classes

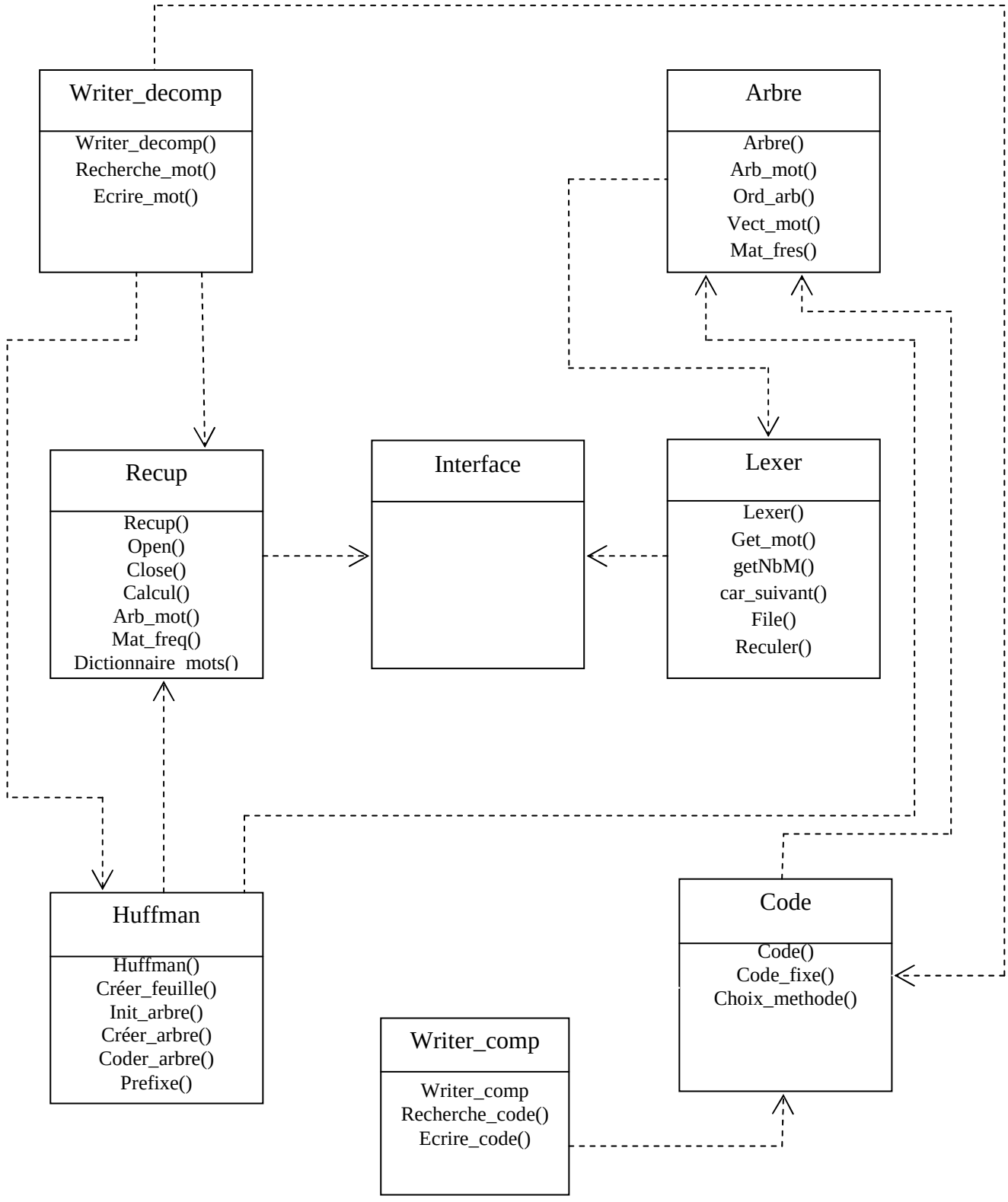


Figure IX : Diagramme de classes

III-1-6-1)-Définition des classes

1)-La classe interface : c'est la classe principale et mère des autres classes.

2)-La classe Lexer (analyseur lexical): permet de lire et de séparer le contenu du fichier source mot par mot.

3)-La classe arbre: crée l'arbre associé au texte et le vecteur des mots (états finaux).

4)-La classe Huffman: Crée l'arbre de Huffman, puis génère le code associé à chaque état final ainsi que sont code préfixe.

5)-La classe code: associe à chaque état final le code correspondant, et permet de choisir la méthode à utilisée (compression et decompression) ?

6)-La classe recup: reconstitue l'arbre des mots ainsi que la matrice des fréquences.

7)-La classe writer_comp : recherche le code associé à l'état final et l'écrit dans le fichier destination.

8)-La classe writer_decomp : lit le fichier caractère par caractère et récupère le mot associé à chaque code et l'écrit dans le fichier destination.

IV)-Conclusion

Dans ce chapitre, nous avons proposé une démarche de modélisation pour développer notre application, en se basant sur la méthode **UML**, en commençant par la spécification des besoins et les divers cas d'utilisation, puis la conception des diagrammes de séquence et de cas d'utilisation en phase d'analyse, et en phase de conception nous avons élaboré les diagrammes d'activité et de classe.

Reste à définir les outils et les langages de programmation qui vont nous aider à mettre en œuvre notre application, ce qui sera l'objet du chapitre suivant, qui est l'implémentation.

Chapitre IV
Implémentation et évaluation

I)-Introduction

Dans le chapitre suivant nous allons passer à la réalisation de notre application en se basant sur les notions et les concepts présentés dans la phase de la conception.

Pour ce faire, notre choix est porté sur un environnement graphique et visuel qui est le Windows, le C++ comme langage de programmation, et un environnement de développement orienté objet qui est le Dev C++.

II)-L'environnement technique de développement :

La réalisation de notre application a été faite sur :

- Un Micro ordinateur **DELL** ayant les caractéristiques suivantes :

Processeur: Intel ® core™ i3-2350M CPU @ 2.30 GHz 2.30 GHz.
Mémoire vive: 4GO.
Système d'exploitation: Windows seven(7) professionnel 64 bits.

- Un Micro ordinateur **HP** ayant les caractéristiques suivantes :

Processeur: Intel ® core™ i3-2310M CPU @ 2.10 GHz 2.10 GHz.
Mémoire vive: 4GO.
Système d'exploitation: Windows seven (7) professionnel 64 bits.

Logiciel utilisé:

- DEV C++ comme environnement de développement.

II-1)-Présentation de DEV C++ :

Le logiciel DEV C++ est un environnement de développement intégré permettant de programmer en C/C++. Il utilise la version MinGW du compilateur GCC (venu du monde du logiciel libre) et permet d'exporter ses projets sous fichiers .dev.

Le compilateur DEV C++ est assez complet. Il comprend entre autre un « répertoire de classes », un « répertoire de fonctions incluses », et un débogueur qui permet de surveiller l'état des variables pendant l'exécution du programme. L'avantage de DEV C++, c'est qu'il est multiplateforme. Il permet de générer des exécutions fonctionnelles sous la majorité des plateformes. L'outil de développement DEV C++ peut être très pratique, il est important de savoir s'en servir efficacement.

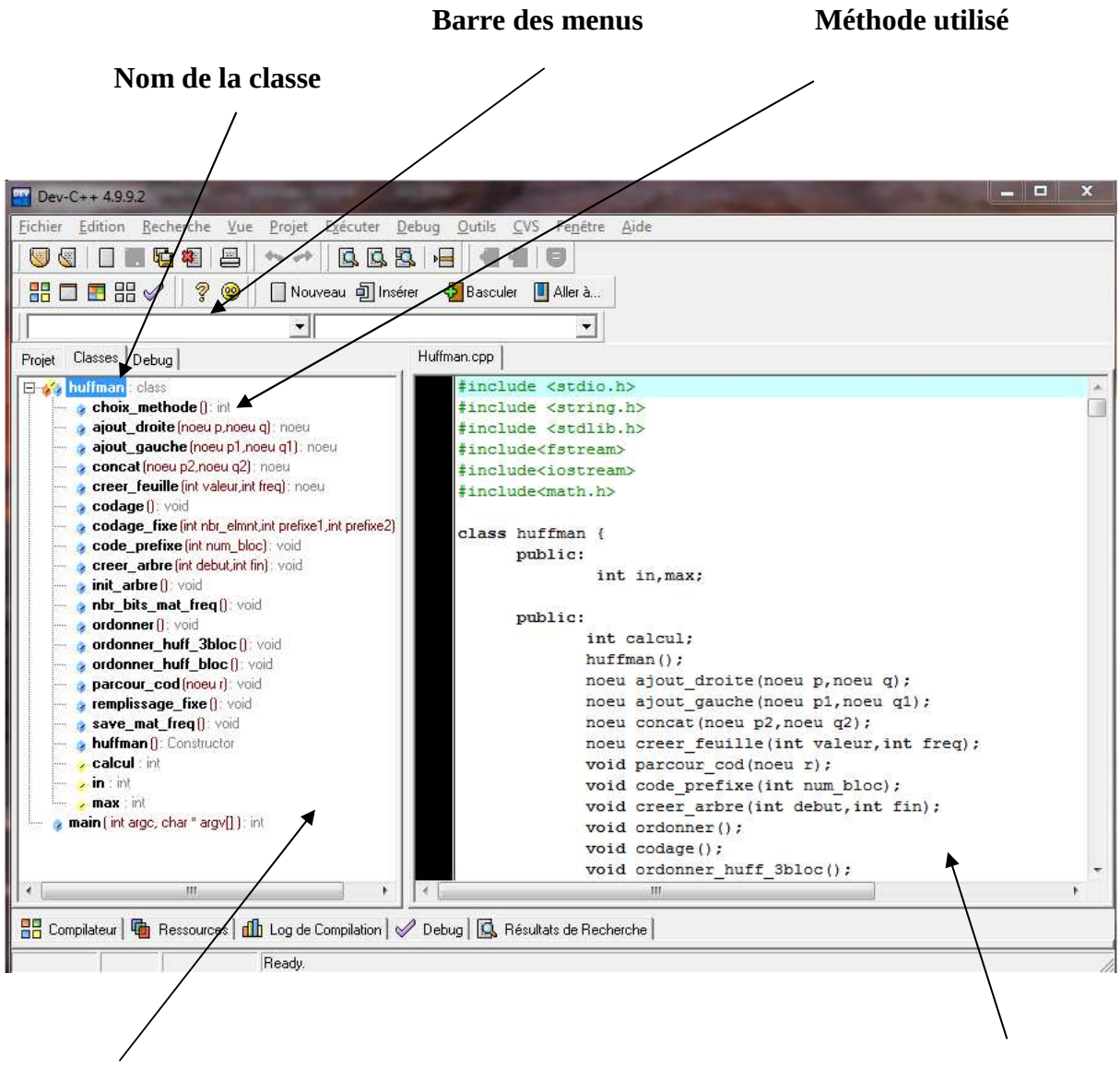


Figure I : Interface graphique DEV C++

II-2)-Présentation du langage de programmation :

Le C++ est l'un des langages de programmation les plus utilisés vu son efficacité. 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 de lui 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 probabilité de leurs programmes au niveau de fichiers sources (pas des exécutables).

Le C++ est un langage orienté objet car il répond aux trois principes fondamentaux : Encapsulations, polymorphisme, et héritage.

En plus, des types simples et complexes du C, on peut définir des classes qui sont des structures évolués dans certains champs (appelées membres) sont des fonctions (appelées méthodes) selon le principe d'encapsulation. Le programmeur doit gérer l'accessibilité aux champs des classes qu'il définit : certains membres peuvent être rendus accessibles directement comme un champ de structure.

Les principaux avantages du C++ sont les suivants :

- Grand nombre de fonctionnalités.
- Facilité d'utilisation des langages objets.
- Portabilité des fichiers source.
- Facilité de conversion des programmes C en C++, en particulier la possibilité d'utilisation de toutes les fonctionnalités du langage C.
- Contrôle d'erreurs accrues.

III)-Présentation de l'application

Dans cette partie, nous allons expliquer le fonctionnement de notre application, ainsi que les commandes nécessaires à sont exécution.

III-1)-Choix du fichier à compresser/décompresser

Pour compressé ou décompressé un fichier il faut que ce fichier et l'exécutable de l'application soient dans le même répertoire, et l'exécution de l'application se passe sur l'invité de commande.

III-2)-La compression

III-2-1)-Syntaxe de la commande de compression

Pour lancer la compression on tape la commande suivante :

Application.exe –c Fichier_source Fichier_destination

Exemple : compresser le fichier bible.txt et le sauvegarder dans le fichier bible.comp

C:\Users\DARISHOP Multimédia\Desktop\ APPLICATION > Application –c bible.txt bible.comp

Où :

C:\Users\DARISHOP Multimédia\Desktop\ APPLICATION > : le chemin de l'exécutable.
Application : le nom de l'exécutable.
-c : pour désigner la compression.
bible.txt : le nom du fichier à compresser.
bible.comp : le nom du fichier compressé.

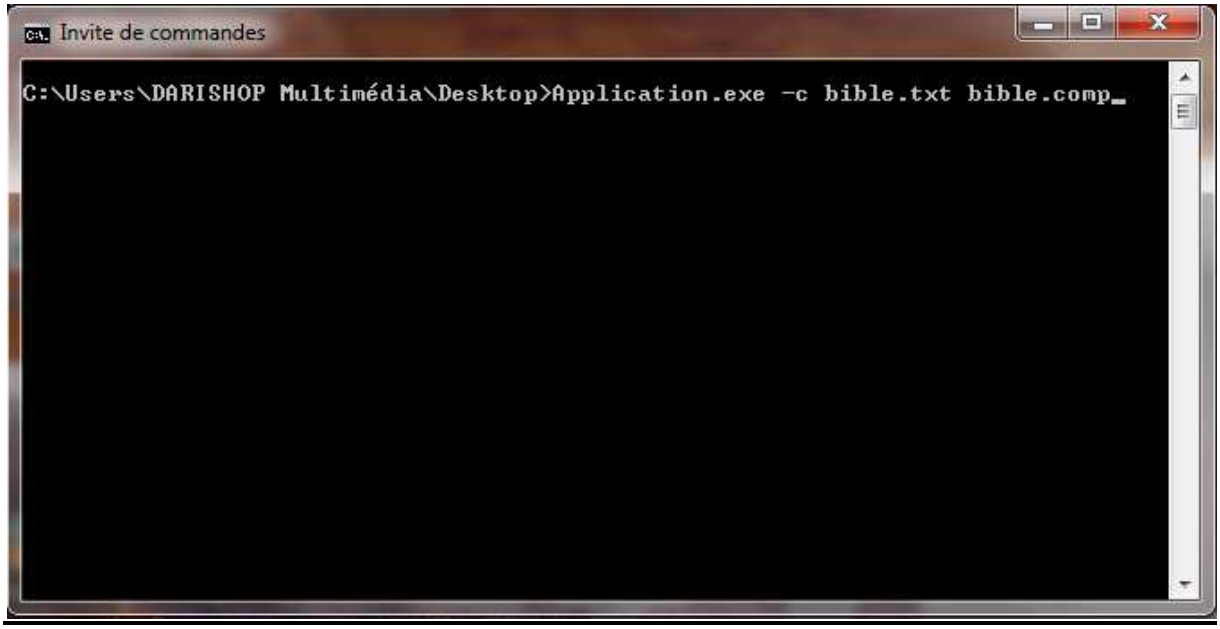


Figure II : Lancement de la compression

III-2-2)-Les résultats de la compression

Après le lancement de la compression nous aurons les résultats suivants :

- ✓ La taille du fichier source.
- ✓ La taille du fichier compressé.
- ✓ Le taux de compression.
- ✓ Le temps de compression.

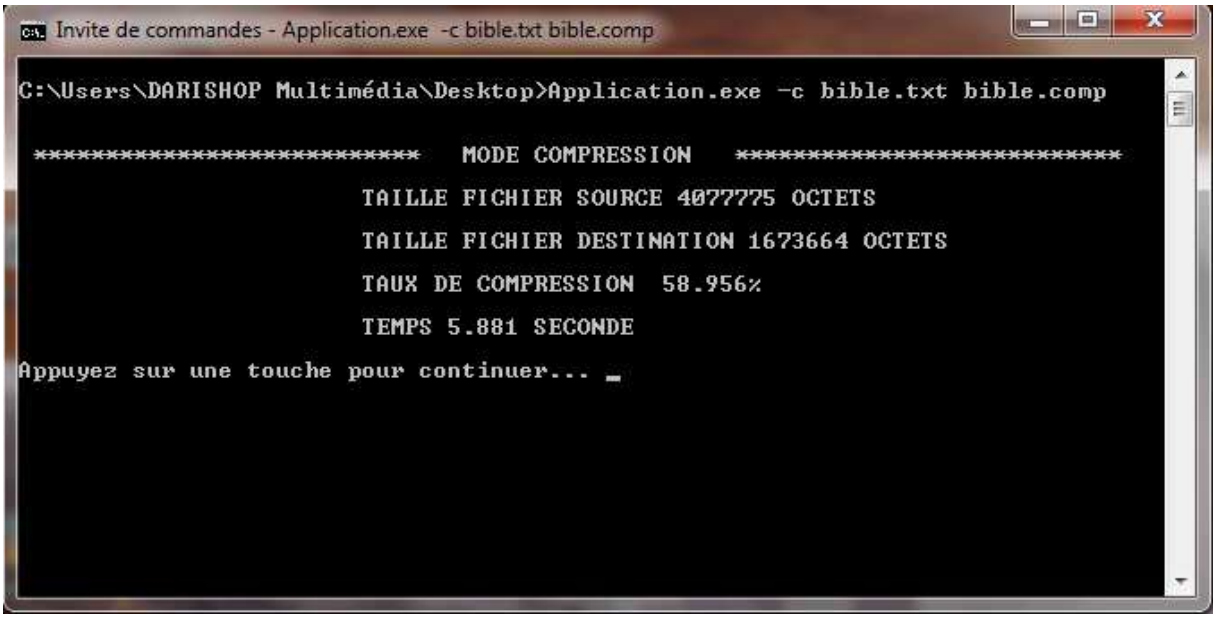


Figure III : Résultats de la compression

III-3)-La décompression

III-3-1)-Syntaxe de la commande de décompression

Pour lancer la décompression en tape la commande suivante :

Application.exe –d Fichier_source Fichier_destination

Exemple : décompresser le fichier bible.comp et le sauvegarder dans le fichier resultat.txt

C:\Users\DARISHOP Multimédia\Desktop\ APPLICATION > Application –d bible.comp resultat.txt

Où :

C:\Users\Desktop\application> : le chemin de l'exécutable.
Application : le nom de l'exécutable.
-d : pour désigner la décompression
bible.comp : le nom du fichier à décompresser.
resultat.txt : le nom du fichier de sauvegarde.

Figure IV : Lancement de la décompression

III-3-2)-Les résultats de la décompression

Après le lancement de la décompression nous aurons le temps de décompression.

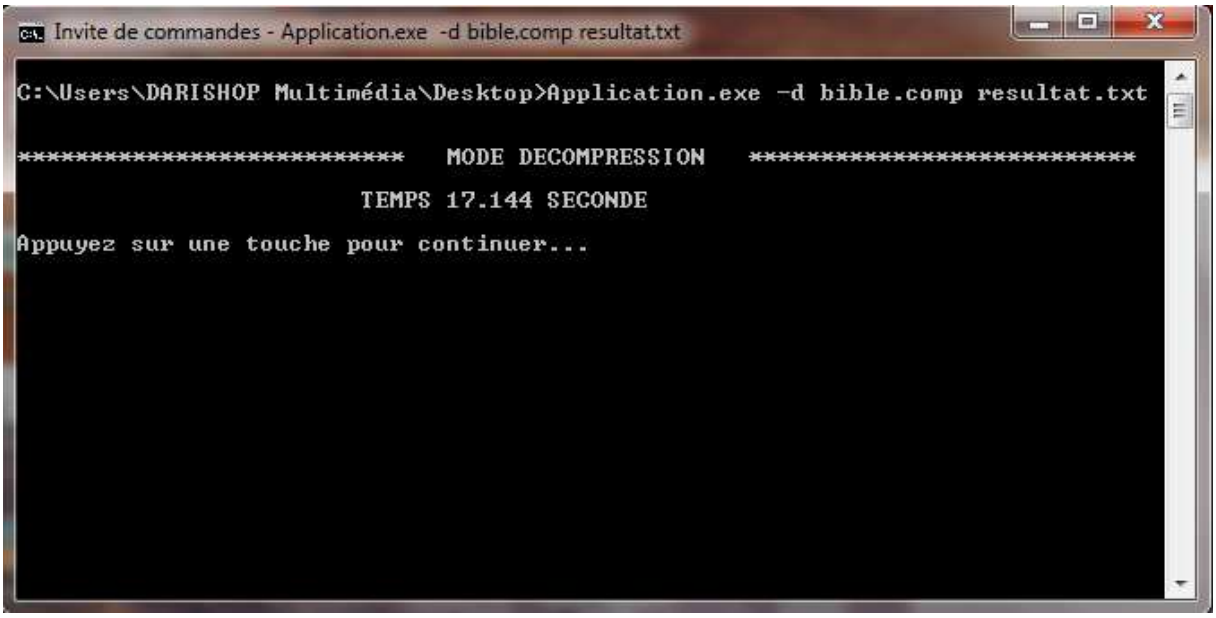


Figure V : Résultats de la décompression

III-4)-Evaluation de l'application

Pour évaluer le taux de compression, nous allons faire des tests sur des fichiers textes, ce taux est calculé par la fonction suivante :

$$\textbf{Taux de compression} = \mathbf{1} - (\frac{\boldsymbol{taille\ fichier\ compressé}}{\boldsymbol{taille\ fichier\ source}} * \mathbf{100})$$

III-4-1)-Tableau des résultats

Nom de fichier	Taille fichier	Taille fichier compressé	Taux de compression	Temps de compression	Temps de décompression
progp.txt	51345	35431	30.994%	0,088	0.393
paper3.txt	47626	47764	-0.290%	0,105	0.359
progl.txt	73890	55145	25.369%	0,127	0.607
paper2.txt	83930	64848	22.736%	0,176	0.531
book1.txt	433047	276182	36.224%	1,06	2.512
book2.txt	626490	331031	47.161%	1,303	3.290
bible.txt	4077775	1673664	58.956%	6,074	17.843

Tableau : Evaluation du taux et temps compression/décompression sur des fichiers textes.

III-4-2)-Présentation du temps et taux de compression

III-4-2-1)-Taux :

La figure suivante représente la variation du taux de compression suivant la taille du fichier à compressé.

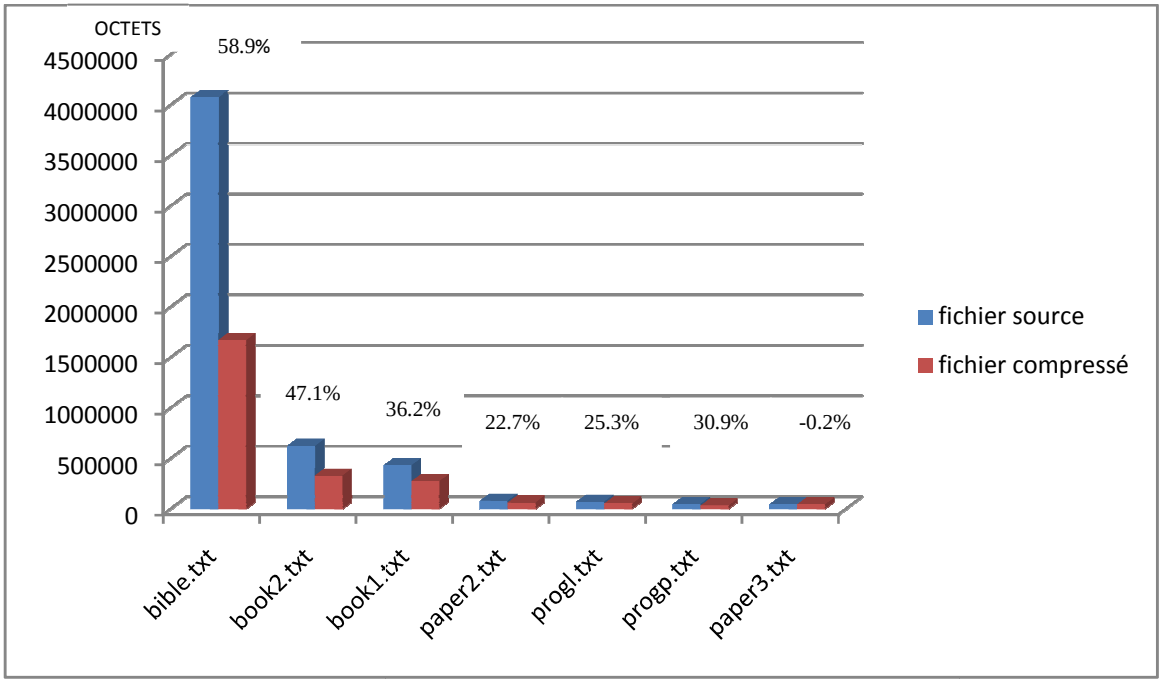


Figure VI : Graphe représentant le taux de compression

Interprétation de la figure :

On remarque dans la figure précédente que le taux de compression augmente selon la taille du fichier. A chaque fois que la taille du fichier est grande le taux de compression augmente.

III-4-2-2)-Temps :

La figure suivante représente la variation du temps de compression suivant la taille du fichier à compressé.

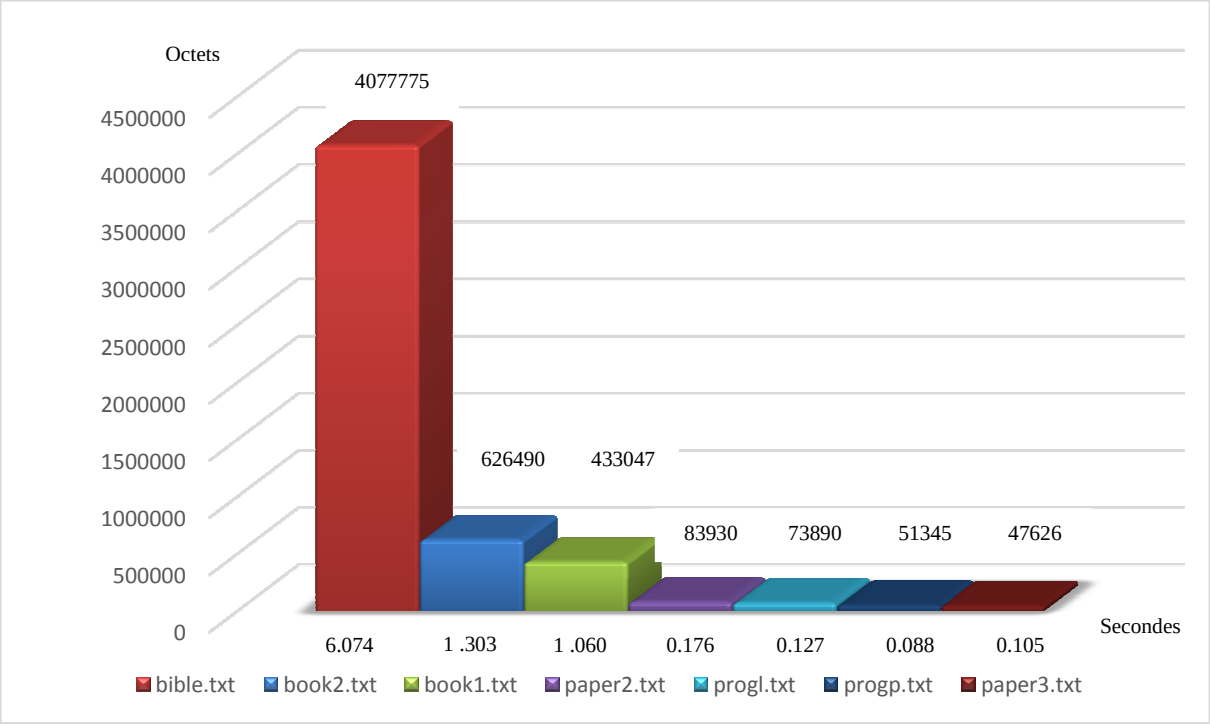


Figure VII : Graphe représentant le temps de compression

Interprétation de la figure :

D'après la figure, précédente on remarque que le temps de compression ne dépend pas toujours de la taille du fichier à compressé. Le temps de compression dépend de la structure du fichier à compressé, c'est-à-dire du nombre de mots différent contus dans ce fichier.

III-4-3)-Présentation du temps de décompression :

La figure suivante représente la variation du temps de décompression suivant la taille du fichier à décompressé.

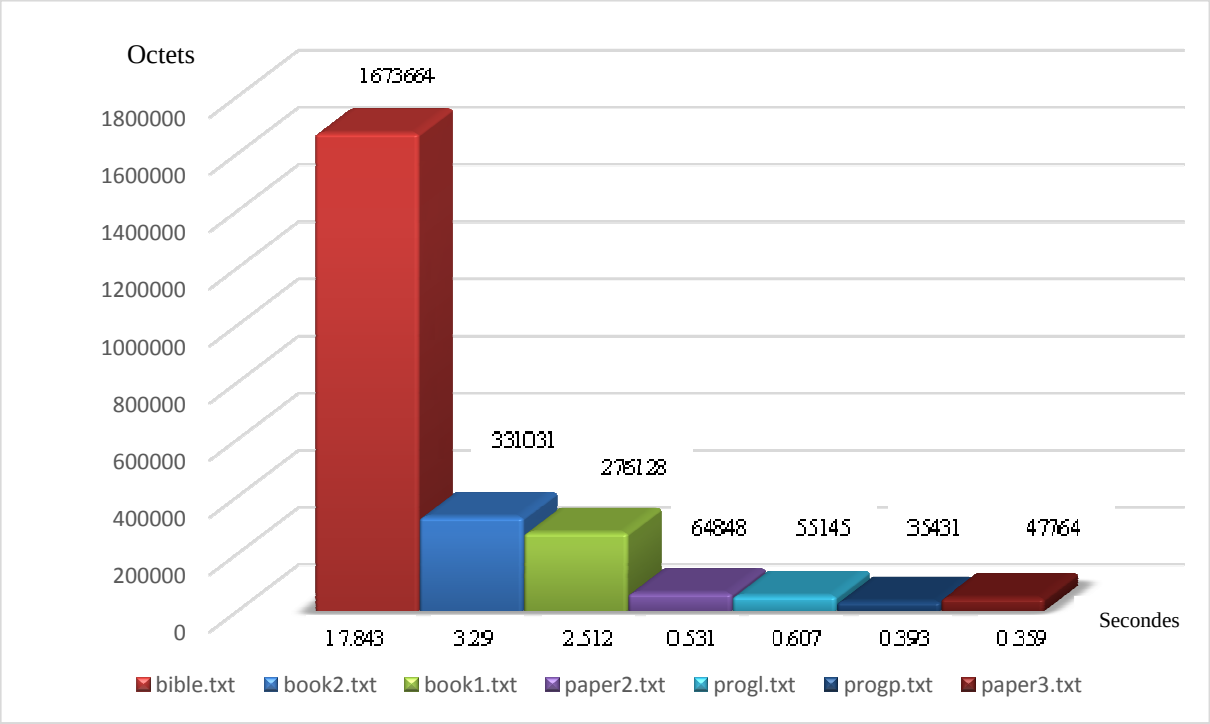


Figure VIII : Graphe représentant le temps de décompression

Interprétation de la figure :

D'après la figure précédente, tout comme la compression, on remarque que le temps de décompression ne dépend pas toujours de la taille du fichier à décompressé. Le temps de décompression dépend de la structure du fichier à décompressé, c'est-à-dire du nombre de mots différent que contient ce fichier.

On remarque aussi qu'il y a une différence de temps par rapport à la compression, et cela est due à la complexité reliée à la décompression, car dans la décompression on a la reconstitution du dictionnaire des mots, la recherche du code bit par bit et l'écriture dans le fichier, tous cela a influencé sur le temps de décompression.

D'après le tableau et les graphes on peut constater que :

- Le taux de compression vari entre -0.290% et 58.956%.
- La moyenne du taux de compression est : 31.59%.
- Le temps de compression vari entre 0.088 Seconde et 6.074 Seconde.
- La moyenne du temps de compression est : 1.276 Seconde.
- Le temps de décompression vari entre 0.393 Seconde et 17.843 Secondes.
- La moyenne du temps de décompression est : 3.647 Secondes.

IV)-Conclusion

Cette partie est consacrée à la présentation de l'environnement de développement, ainsi que le langage de programmation utilisé, et le fonctionnement de notre application. Par la suite nous avons fait des tests sur un ensemble de fichier fréquemment utilisés dans la compression de données pour comparer les techniques de compression entre elles, et cela précédé par un certain nombre de graphe pour mieux illustré la différence qui existe entre ces fichiers.

Conclusion générale

La compression de données est appelée à prendre un rôle encore plus important en raison du développement des réseaux et du multimédia. Plusieurs méthodes de compression de données ont vu le jour, cette variétés de méthodes est due à la diversité des types de données ciblés (texte, image, audio etc).

Notre travail avait comme objectif l'implémentation et l'évaluation d'une nouvelle méthode de compression de données sans perte, qui utilise la méthode de Huffman basée sur un traitement par mots, fusionnée avec le codage fixe. Pour ce faire il est nécessaire de comprendre et de connaitre divers concepts dans le monde de la compression de données et de la programmation.

A la fin de notre travail, nous avons constaté que cette nouvelle méthode de compression de données donne de bons résultats soit par rapport au taux et au temps de compression, soit par rapport au temps de décompression pour les fichiers texte.

Cette expérience nous a permis d'acquérir beaucoup de connaissance dans la compression de données qui est un domaine très vaste et dans la programmation orienté objet C++.

Nous précisons que les différentes parties de notre application, peuvent être optimisé pour donner de meilleurs résultats, surtout par rapport au temps de décompression pour en faire une application symétrique.

Annexes

I)-UML

I-1)-La modélisation avec UML

UML (Unified Modeling language) (**Langage de modélisation objet unifié**) :

Est né de la fusion de trois méthodes qui ont le plus influencé la modélisation objet au milieu des années 90 OMT, BOOCH et OOSE.

Issu du terrain et fruit d'un travail d'experts reconnus, UML est le résultat d'un large consensus, de très nombreux acteurs industriels de renom ont adopté UML et participent à son développement.

I-2)-Le processus unifié

Le processus unifié est un processus de développement logiciel, il regroupe les activités à mener pour transformer les besoins d'un utilisateur en système logiciel.

Caractéristiques essentielles du processus unifié :

- Le processus unifié est à base de composants.
- Le processus unifié utilise le langage UML (ensemble d'outils et de diagramme).
- Le processus unifié est piloté par les cas d'utilisation.
- Centré par l'architecture.
- Itératif et incrémental.

I-3)-Les éléments structurels

I-3-1)-La classe

Une classe est une description abstraite condensée, d'un ensemble d'objets et domaines de l'application. Les classes sont représentées par des rectangles compartimentés, le premier compartiment contient le nom de la classe, le deuxième compartiment contient les attributs, le troisième compartiment contient les opérations et le quatrième facultatif représente les responsabilités.

Notons que les attributs et les opérations peuvent être d'une visibilité publique, protégée ou privée.

Exemple :

La figure suivante illustre ces différents comparatifs avec les visibilités suivantes :

Name: publique, name2: protégée, name3 : privée, name4 : rien.

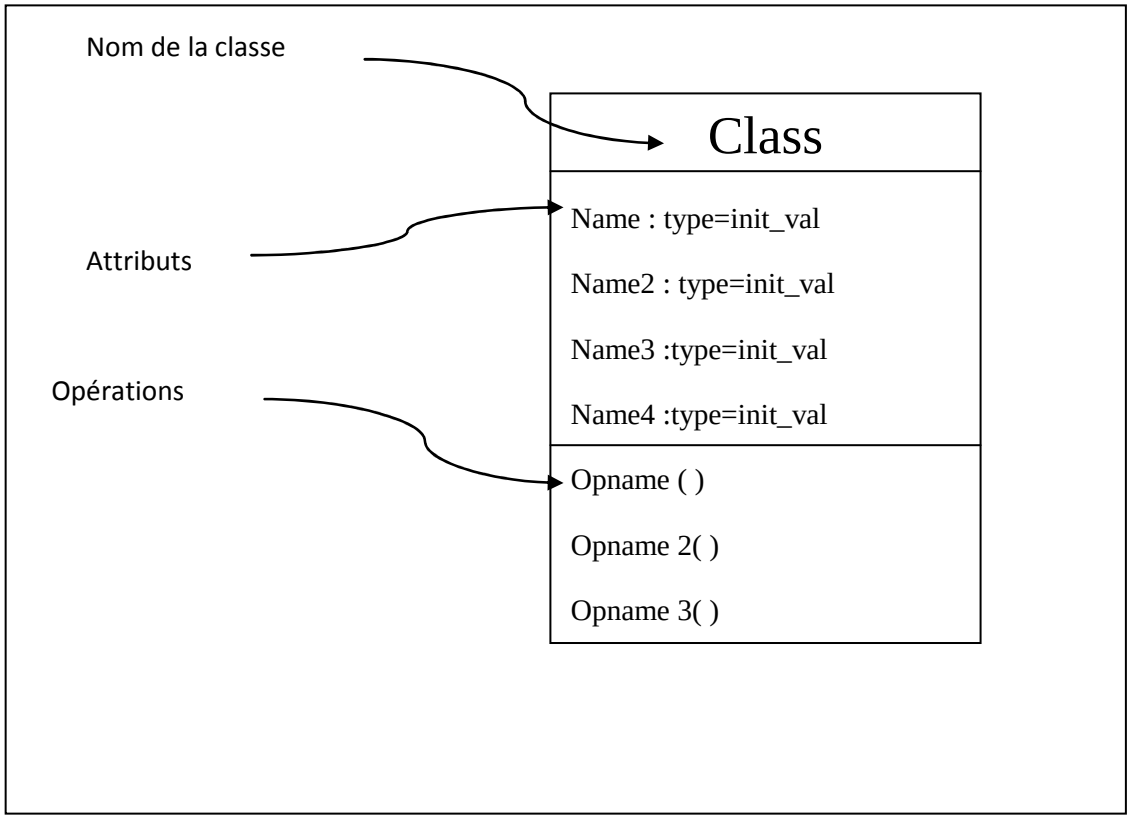


Figure I : Schéma représentatif d'une classe

I-3-1-1)-Les classes-associations :

Il est possible de représenter une association par une classe pour ajouter, par exemple, des attributs et des opérations dans l'association. Une classe de ce type, appelée classe-associative ou classe-association, possède à la fois les caractéristique d'une classe et d'une association, et peut à ce titre participer à d'autres relation dans le modèle. La notation utilise une ligne pointillée pour attacher une classe à une association.

Dans la figure suivante, l'association entre les classes A et B est représenté par la classe C.

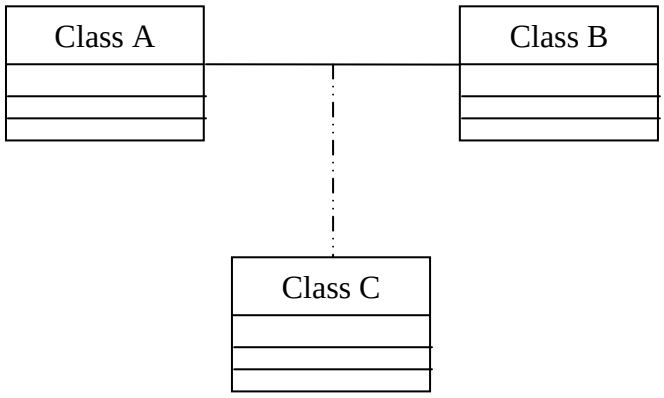


Figure II : Schéma représentatif d'un classe-association

I-3-2)-Les cas d'utilisation

Formalisés par Ivar Jacobson, les cas d'utilisation décrivent sous la forme d'action et de réaction le comportement d'un système du point de vue de l'utilisateur, et permettent de définir les limites du système et les relations entre le système et l'environnement, c'est l'image de fonctionnalité du système, déclenchée en réponse à la simulation d'un acteur externe, les cas d'utilisation sont représentés par des ellipses contenues par le système.

I-4)-Les éléments comportementaux

Ils sont les parties dynamiques des modèles UML, ils comprennent les interactions et les automates à états finis.

I-4-1)-Les interactions

Une interaction exprime le comportement qui résulte de la collaboration d'un groupe d'instances.

Elle peut être visualisée selon le point de vue temporel (diagramme de séquences) ou selon le point de vue de l'espace (diagramme de collaboration).

La figure suivante schématise une interaction visualisée selon le point de vue temporel.

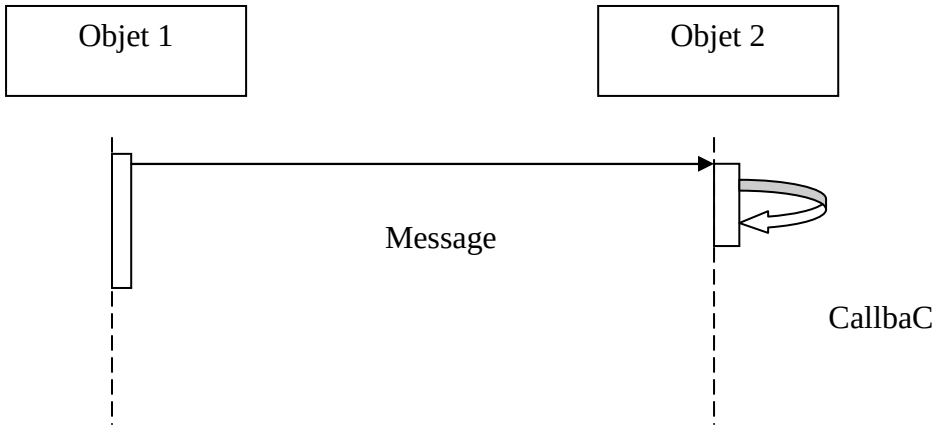


Figure III : Schéma représentatif d'une interaction

I-5)-Les éléments de regroupement

Ils sont les parties organisationnelles des modèles UML, ils comprennent les paquetages.

I-5-1)-Les paquetages :

Ils regroupent les trois types d'éléments précédents, ils sont purement conceptuels c'est-à-dire qu'ils n'existent que lors de la phase de développement.

Ils offrent un mécanisme général pour la partition des modèles et le regroupement des éléments. Un paquetage est représenté en général par un dossier étiquette, et contient seulement son nom, mais parfois sont contenu.

Paquetage

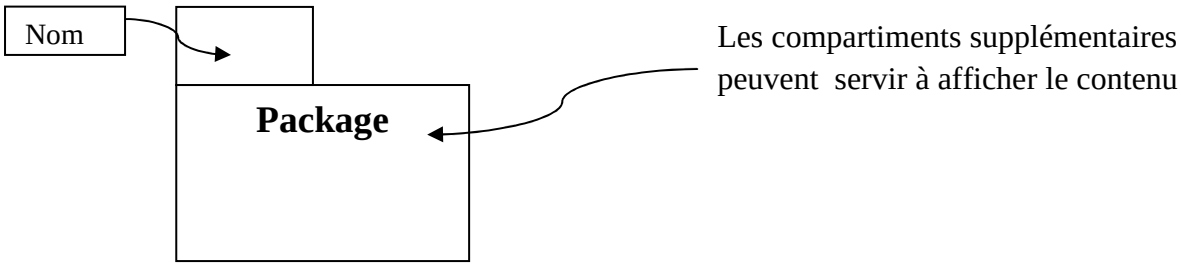


Figure IV : Schéma représentatif d'un paquetage

I-5-2)-Les éléments d'annotation :

Ils sont les parties explicatives des modèles UML ils comprennent les notes.

Les notes : ce sont les commentaires qui sont attachés à un ou plusieurs éléments de modélisation. Une note est représentée par un rectangle écorné qui contient un commentaire textuel ou graphique. Comme le montre la figure suivante :

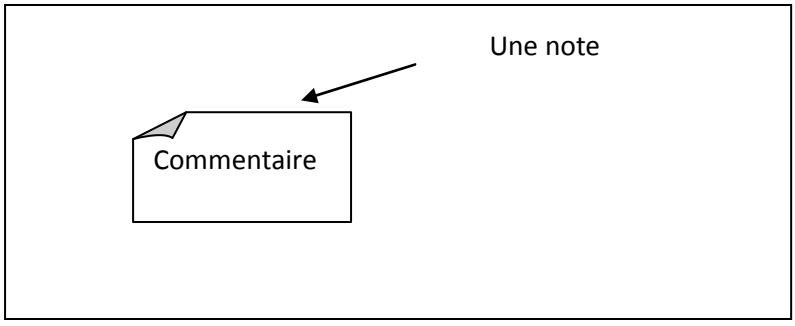


Figure V : Schéma représentatif d'une note

I-6)-Les relations

I-6-1)-Dépendance :

Elle représente un lien de dépendance entre deux(2) éléments de la modélisation et dont la modification d'un élément (élément indépendant) peut affecter la sémantique de l'autre élément (l'élément dépendant). Elle est représentée par un trait en pointillé.

Source ----------> cible

Figure VI : Schéma représentatif d'une relation de dépendance

I-6-2)-Association

Elle représente une relation entre classes d'objets. Elle exprime une connexion sémantique bidirectionnelle entre les classes. En UML, elle est représentée par un trait plein entre deux classes. L'extrémité d'une association est appelée rôle, chaque rôle porte une

indication de multiplicité qui montre combien d'objets de la classe considérée peuvent être liés à un objet de l'autre classe.

I-6-3)-La généralisation

C'est une relation de spécification / généralisation entre une classe et une ou plusieurs autres classes partageant un sous-ensemble commun d'attributs et/ou de méthodes.

UML emploie le terme de généralisation pour désigner la relation de classification entre un élément plus général (parent) et un élément plus spécifique (enfant). La généralisation peut s'appliquer aux classes, cas d'utilisation, et paquetages.

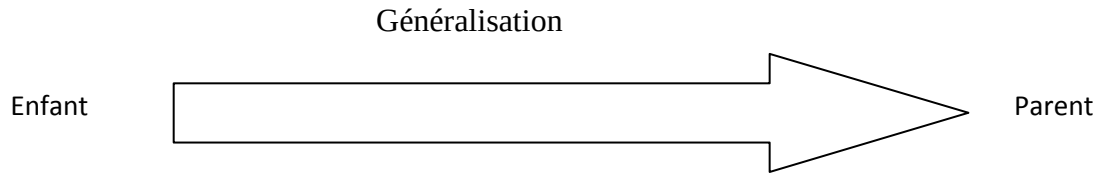


Figure VII : Schéma représentatif d'une généralisation

I-7)-Les diagrammes d'UML

Un diagramme est la représentation graphique d'un ensemble d'éléments, par un graphe relié de sommets (éléments) et d'arcs (relation).

I-7-1)-Les vues statique de l'UML

I-7-1-1)-Le diagramme de cas d'utilisation

Les cas d'utilisation ne sont pas seulement un simple outil de spécification des besoins du système, mais ils vont complètement guider le processus de développement à travers l'utilisation de modèles basés sur l'utilisation du langage UML, les cas d'utilisation garantissent la cohérence du processus de développement du système. S'il est vrais que les cas d'utilisation guident le processus de développement, ils ne sont pas sélectionnés de façon isolée, mais doivent absolument être développés en tandem avec l'architecture du système.

I-7-1-2)-Le diagramme de classe

Le diagramme de classe exprime de manière générale la structure statique d'un système, en termes de classes et de relation entre ces classes.

Une classe permet de décrire un ensemble d'objets (attributs et comportement), tandis qu'une relation ou association permet de faire apparaitre des liens entre ces objets.

On peut donc dire :

- Un objet est une instance de classe.
- Un lien est une instance de relation.

Le diagramme de classe est un modèle permettant de décrire de manière abstraite et générale les liens entre objets.

a)-Les classes frontières (interface) : classes qui servent à modéliser les interactions entre le système et ces acteurs.

b)-Les classes contrôles : classent qui servent à représenter la coordination, le séquencement, les transactions et le contrôle d'autres objets.

c)-Les classes entité : classent qui servent à modéliser les informations durables et persistantes.

I-7-1-3)-Le diagramme de composants

Les diagrammes de composant permettent de décrire l'architecture physique et statique d'une application en termes de modules (fichier source, librairies, exécutable, etc…)

Ils montrent la mise en œuvre physique des modèles de la vue logique avec l'environnement de développement.

Les dépendances entre composants permettent notamment d'identifier les contraintes de compilation, et de mettre en évidence la réutilisation des composants, qui peuvent être organisés en paquetage, qui définissent des sous-systèmes. Ces derniers organisent la vu des composants du système, et permettent de gérer la complexité, par encapsulation des détails d'implémentation.

I-7-1-4)-Diagramme de déploiement

Les diagrammes de déploiement montrent la disposition physique des matériels qui composent le système et la répartition des composants sur ces matériels.

- Les ressources matérielles sont représentées sous forme de nœuds.
- Les nœuds sont connectés entre eux, à l'aide d'un support de communication.
- La nature des lignes de communication et leurs caractéristiques peuvent être précisées.
- Les diagrammes de déploiement peuvent montrer des instances de nœuds, ou des classes de nœuds.
- Les diagrammes de déploiement correspondent à la vue de déploiement d'une architecture logicielle.

I-7-2)-Les vues dynamique d'UML

I-7-2-1)-Diagramme de séquence

Les diagrammes de séquence permettent de représenter des collaborations entre objets selon un point de vue temporel, on y met l'accent sur la chronologie des envois de messages.

Contrairement aux diagrammes de collaboration, on n'y décrit pas le contexte ou l'état des objets, la représentation se concentre sur l'expression des interactions.

Les diagrammes de séquence peuvent servir à illustrer un cas d'utilisation. L'ordre d'envoi d'un message est déterminé par sa position sur l'axe verticale du diagramme, la disposition des objets sur l'axe horizontal n'a pas de conséquence pour la sémantique du diagramme.

I-7-2-2)-Diagramme d'états-transitions

Ce diagramme sert à représenter des automates d'états finis, sous forme de graphes d'états, reliés par des arcs orientés qui décrivent les transitions.

Les diagrammes d'états-transitions permettent de décrire les changements d'états d'un objet ou d'un composant, en réponse aux interactions avec d'autres objets/composant ou avec des acteurs. Un état se caractérise par sa durée et sa stabilité, il représente une conjonction instantanée des valeurs des attributs d'un objet.

Une transition représente le passage instantané d'un état vers un autre, elle est déclencher par un événement. En d'autres termes c'est l'arriver d'événements qui conditionnent la transition.

Les transitions peuvent aussi être automatiques, lorsqu'on ne spécifie pas l'événement qui la déclenche. En plus de spécifier un événement précis, il est aussi possible de conditionner une transition, à l'aide de grades, qui s'agit d'expressions booléennes, exprimées en langage naturel.

Les diagrammes de séquence et les diagrammes d'états-transitions, sont les vues dynamiques les plus importantes d'UML.

I-7-2-3)-Diagramme d'activités

UML permet de représenter graphiquement le comportement d'une méthode ou le déroulement d'un cas d'utilisation, à l'aide de diagramme d'activités.

Une activité représente une exécution d'un mécanisme, et le déroulement d'étapes séquentielles.

Le passage d'une activité vers une autre est matérialisé par une transition.

Les transitions sont déclenchées par la fin d'une activité et provoquent le début immédiat d'une autre.

En théorie, tous les mécanismes dynamiques pourraient être décrits par un diagramme d'activités, mais seuls les mécanismes complexes ou intéressants méritent d'être représentés.

I-7-2-4)-Diagramme de collaboration

Le diagramme de collaboration permet de mettre en évidence les interactions entre les différents objets du système étudié, ainsi que les messages qu'ils échangent entre eux.

A l'aide du diagramme de collaboration, nous illustrons donc l'interaction entre objets en créant des liens entre ces objets, et en associant des messages a ces liens. Le nom d'un message doit évoquer l'intention de l'objet appelant lors de l'interaction avec l'objet associé.

Un diagramme de collaboration permet de décrire les interactions entre objets intervenant dans la réalisation d'un scénario d'un cas d'utilisation.

III-Quelques algorithmes utilisés :

III-1)-Algorithme parcoure fichier et création de l'arbre

```
Tant que (Non_fin_de_fichier)

Début
  Car=mot (fichier) ;
  Tant que (taille_mot >= 0)
    Si (arbre [ligne] [(entier)car[i]] < > 0)

    Début
        val=arbre [ligne][(entier)car[i]]) ;
        Ligne=val ;
        i++ ;
        taille_mot-- ;
    Fin Si

    Sinon
    Tant que (taille_mot >= 0)

    Début
        Compteur++ ;
        Arbre [ligne][(entier)car[i]] = compteur ;
        Ligne=compteur ;
        Taille-- ;
    Fin Tant que

Fin.
```

III-2)-Alorithme de récupération des états finaux

```
Début

Tant que (Non fin de fichier)

Début

    Taille_mot=longueur (mot) ;

    Tant que (taille_mot>0)

     Début

        Indice=arbre_ordonner [indice] [entier(car[i])] ;

        Taille_mot-- ;

         i++ ;

      Fin tant que
```

```
    Etat_final[j] =indice ;
    i++;
  Fin Tant que;
Fin.
```

III-3)-Algorithme création matrice des fréquences

```
Début
Pour (i=0 jusqu'à taille (etat_final [ ]))
Début
   Si (etat_final[i] existe dans (mat_freq) alors
   Debut
      Récupérer (indice (mat_freq)) ;
      Mat_freq [indice] [1] ++ ;
    Fin si ;
    Sinon
      Mat_freq [j] [0] =etat_final [i];
      Mat_freq [j] [1] =1;
      J++
     Fin Sinon
Fin pour
Fin.
```

III-4)-algorithme créer arbre de Huffman :

```
Début
Arbre.val=0 ;
Pour (i=0 jusqu'à 256)
Début
      Valeur=dépiler (file) ;
      Si (valeur<dernier_fils)
```

```
        Si (valeur + val_suivante < dernier_fils)


        Début
              Arbre_temp.gauche.etat = élément_état [i] ;
              Arbre_temp.droite.etat = élément_état [i+1] ;
              Arbre_temp.freq=valeur+val-suivante ;
              Arbre.droite=arbre ;
              Arbre.gauche=arbre_temp ;
              Arbre.freq=arbre.freq+arbre_temp.freq ;
               i++ ;
        Fin si ;
       Sinon
       Début
            Arbre.gauche.etat = element[i] ;
            Arbre.droite=arbre ;
            Arbre.freq=arbre.freq+valeur ;
       Fin sinon;
    Sinon
       Début
            Arbre.droite.etat = element[i] ;
            Arbre.gauche=arbre ;
            Arbre.freq=arbre.freq+valeur ;
       Fin sinon ;
  i++ ;
Fin pour;


Parcour(arbre)
Si (arbre.gauche)
Arbre.gauche.code=arbre.code+ajout (0) ;
```

Si (arbre.droite)

Arbre.doite.code=arbre.code+ajout (1) ;

Fin.

III-5)-Algorithme choix méthode :

Début

Si (nombre_ligne(matrice_frequence) > 768)

Création de 3 arbres de HUFFMAN et le reste c'est le codage fixe ;

Sinon

Si (256 < nombre_ligne(matrice_frequence) < 768)

Création d'un arbre de HUFFMAN et le reste c'est le codage fixe ;

Sinon

Codage fixe ;

Fin.

III-6)-Algorithme codage fixe

Début

Nombre_bits = log_2 (taille (matrice_fréquence)

Pour (i=0 jusqu'à taille (matrice_fréquence))

Début

Code[i][0] = matrice_fréquence[i][0] ;

Code[i][1] = code_binaire[i] ;

Fin pour ; Fin.

III-7)-Algorithme récupération arbre des mots :

Début

Ligne=0 ;

nbr_etats= Lire (fichier_compresse) ;

Tant que (nbr_états>0)

Début

```
Nbr_car=lire (fichier_compresser) ;
Tant que (nbr_car>0)

Début
        Car=lire (fichier) ;
        Arbre [ligne] [entier (car)]=compteur ;
        Compteur++ ;
        Nbr_car-- ;
Fin tant que;
Ligne++ ;
Nbr_état-- ;
Fin tant que ;
Fin.
```

III-8)-Algorithme récupération matrice des fréquences :

```
Début
 Nbr_ligne_mat=lire (fichier_compressé) ;
J=0 ; k=0 ;
Pour (i=0 jusqu'à nbr_ligne_mat*2)
 Début
    Tant que (nombre non recupérer)
    Début
       Car=Lire (fichier compressé) ;
      Nombre=nombre+entier (nombre) ;
   Fin tant que;
   Mat_freq[k][j]=nombre ;
   J++ ;
  Si (j==2) alors
  Début
     J=0 ;
```

```
        k++ ;
      Fin si;
    Fin pour;
Fin.
```

III-9)-Algorithme reconstitution dictionnaire des mots

```
Début
  Pour (i=0 jusqu'à nombre_ligne)
  Pour (j=0 jusqu'à 255)
  Si (arbre[i][j] < > NULL)
  Début
        Indexe_mots[arbre[i][j]] = i ;
        Indexe_lettre[arbre[i][j]] = caractére (j);
  Fin Si
  Pour (i=0 jusqu'à nombre_etats_finaux)
  Début
        Nombre_lettre = 0 ;
        Etat_final = mat_freq [i][0] ;
        Indexe_final = indexe_mots[etat_final];
        Text[i][0] = etat_final ;
        Tant que (etat_final > 0)
        Début
          Mot_inverse[nombre_lettre] = indexe_lettre[etat_final] ;
          Nombre_lettre ++ ;
          Etat_final = indexe_final ;
          Indexe_final = indexe_mot[etat_final] ;
        Fin Tant que
    Text[i][1] = inverse(mot_inverse);
  Fin Pour
Fin .
```

III-10)-Algorithme création buffer :

```
Début
   Si (non fin de fichier)
   Début
      Car=lire (fichier compressé) ;
      Buff=binaire (car) ;
   Fin ;
Fin.
```

III-11)-Algorithme lecture buffer

```
Début
  Si (buffer vide)
    Créer_buff() ;
  Retourner (buffer) //bit par bit
Fin.
```

III-12)-Algorithme recherche code et écriture dans fichier :

```
Début
   Tant que (non fin de fichier)
   Début
      Tant que (mot non trouvé)
        Code=lire buffer () ;
      Recherche (dictionnaire des mots, code) ;
      Ecrire (mot, fichier destination) ;
  Fin tant que ;
Fin.
```

Bibliographie

[1] Pereira Vincent – Hacaulit Vincent – Leprette Frank compression de données.

[2] Nicolas Sendrier Théorie de l'information.

[3] Marie-pierre Béal – Nicolas sendrier (note de cours) Novembre 2012.

[4] Souhel meshoul Compression de données (cours).

[5] S.Maadi – Y.pneveyre – C.Lamberg Compression de données avec perte.

[6] Pierre Jouvelot Compression de données texte, image, son 2006.

[7] Jean-François Pillou –compression de données Juin 2004.

[8] Steven Pigeon – contribution à la compression de données 2001.

[9] Julien Fayolle – Compression de données sans perte et analyse combinatoire Mars 2006.

[10] Alexandre Pauchet Modélisation des systèmes complexes, introduction aux automates à états fini 2011.

[11] Florent Hivert – Les arbres.

[12] Andrei Cubitchi – Miranda Nafornita – Alexendre Idar - Algorithmes et techniques de compression 2000.

[13] IREM de Lyon – Parcour des arbres.

[14] Thierry Lecroq – Compression méthode de HUFFMAN.

[15] Vincent Vajnouszki – Information et quantité d'information.

[16] ESPCI – Entropie de Shannon, théorie de l'information et compression de données.

[17] https://www.wolframscience.com/reference/notes/1069b