Compte rendu de TP de compression

Binôme	Morvan Lassauzay & Sacha Kierbel
Diplôme	Licence informatique 3éme année
Année universitaire	2014/2015

Travail réalisé

Grille de déclaration

	Fixe	Variable	RLE	Huffman	LZW
Compression/Décompression	100%	100%	100%	100%	100%
Mesure sur échantillon	100%	100%	100%	100%	100%

Fonctionnement général

Pour exécuter les programmes ont utilise la commande : ./main nom_fichier_de_données .
A noter que pour choisir le codage que l'on souhaite effectuer il suffit de retirer les commentaires autour du code de celui-ci dans le fichier « main.cpp ».
Pour effectuer le décodage, le décodeur a besoin d'informations complémentaires (comme la table des codes, etc...) en plus des données codées. En pratique un échange préalable peut avoir été effectué pour envoyer ces données ou établir un accord autour d'une structure d'envoi permettant au décodeur de récupérer correctement ces informations. Dans notre cas ces données sont comprises dans la classe et définies par le codeur, elles seront donc directement accessible par le décodeur.

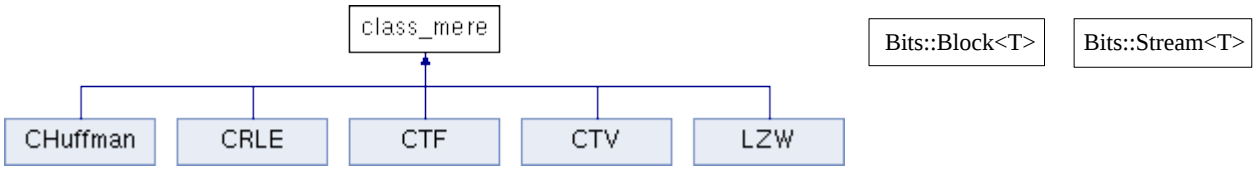
Fonctionnalités supplémentaires implémentées :

- Excepté pour le codage à taille fixe, l'alphabet est calculé à partir d'une première lecture des données. Le nombre de symboles doit cependant être connu à l'avance pour permettre l'allocation des structures utiles au codage et décodage. Il aurait également été possible d'effectuer une allocation dynamique des structures après avoir compté le nombre de symboles différents grâce à une première lecture des données, mais dans notre cas cela ne semble pas nécessaire ni réellement avantageux.

Rapport

A. Structure logicielle générale

Chacun des codages dispose de sa propre classe et de ses propres fichiers (.cpp et .h). Néanmoins les différents codages disposent de caractéristiques communes. Ils héritent donc tous d'une classe mère qui dispose d'une méthode permettant de lire les données à partir d'un des fichiers fourni, d'une méthodes permettant d'afficher la taille des données codées et non codées, des prototypes des méthodes virtuelles d'encodage et de décodage. Les classes décrites par les fichiers « BitStream.cpp » et « BitStream.h » seront également utilisées par l'ensemble des codages.



B. Codage à taille fixe

1. Structure logicielle

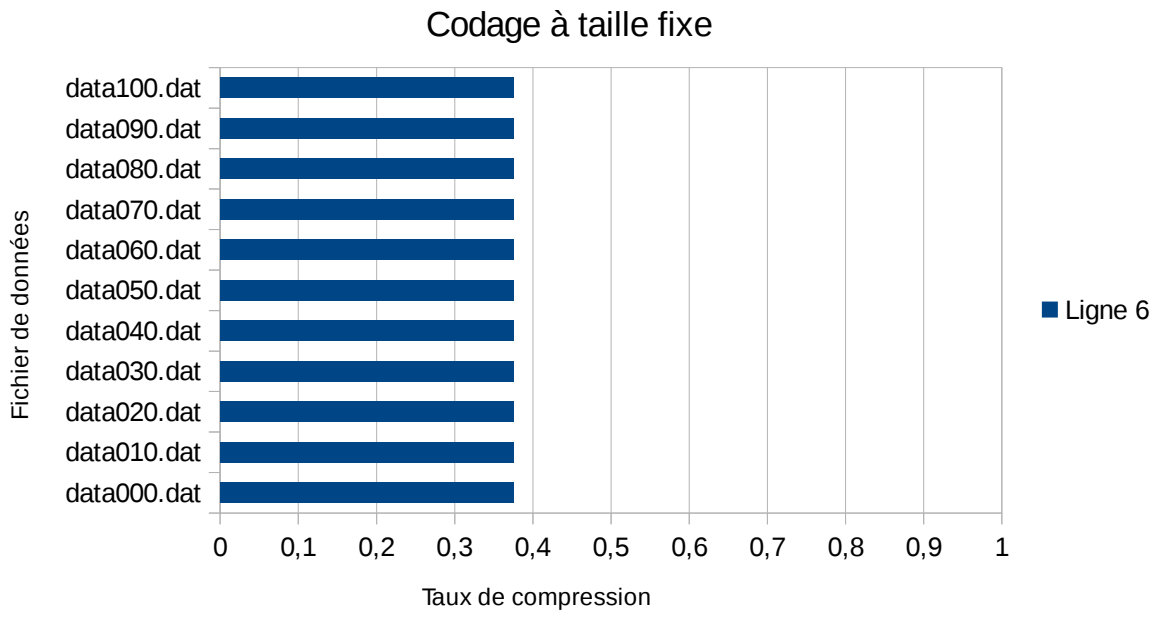
Pour le codage à taille fixe la classe « CTF » a été utilisée. Cette classe est décrite par les fichiers « CTF.h » et « CTF.cpp » et se contente d'implémenter les méthodes virtuelles « encode » et « decode » héritées de la classe mère. Voir diagramme de la partie A.

2. Compression / décompression

Notre implémentation nécessite de connaître à l'avance le nombre de symboles pour déterminer le nombre de bits utilisés pour coder chaque symboles (define BITLEN dans CTF.h). Le programme est fonctionnel mais nécessite des symboles ayant une valeur plus grande que celle du « A » dans la table ascii. Un suivi du codage est possible en retirant les portions de code mises en commentaire dans le fichier « CTF.cpp ».

3. Mesure de compression

Les mesures de compression ont été effectuées uniquement à partir du nombre de bits résultants du codage des données. Ici le code de chaque symbole étant calculé directement à partir du symbole, seul le nombre de bits utilisé pour coder les symboles, soit environ 8 bits, constituerait l'entête à transmettre au décodeur pour qu'il puisse effectuer le décodage seul.



B. Codage à taille variable

1. Structure logicielle

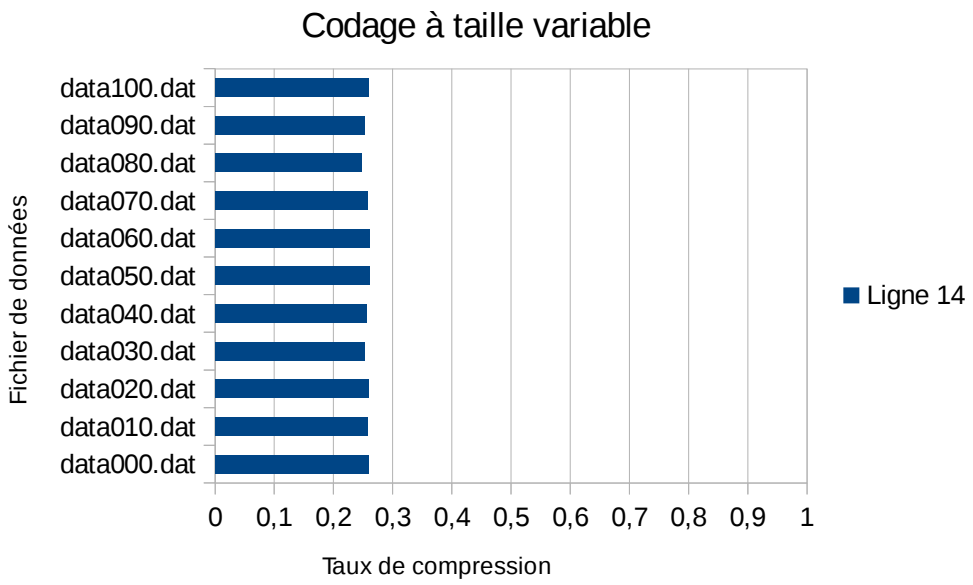
Pour le codage à taille variable la classe « CTV » a été utilisée. Cette classe est décrite par les fichiers « CTV.h » et « CTV.cpp ». Elle implémente bien entendu les méthodes virtuelles de la classe mère mais également d'autres méthodes qui lui sont propres. Voir diagramme de la partie A.

2. Compression / décompression

Notre implémentation établi la liste des symboles grâce à une première lecture des données. Cependant le nombre de symboles (define NB_SYMBOLES_CTV dans CTV.h) doit être renseigné à l'avance pour permettre le bon déroulement du codage. Chaque symbole est codé par un certain nombre de « 1 » selon sa fréquence auxquels ont ajoute « 0 » faisant office de délimiteur.

3. Mesure de compression

Les mesures de compression ont été effectuées uniquement à partir du nombre de bits résultants du codage des données. Pour que le décodeur puisse effectuer le décodage seul il faudrait alors lui transmettre en plus des données codées, une entête contenant la table des symboles triés par ordre de fréquence ainsi que le nombre de symboles dans l'alphabet si besoin, soit au total 72 bits pour un alphabet de 8 symboles.



C. Codage RLE

1. Structure logicielle

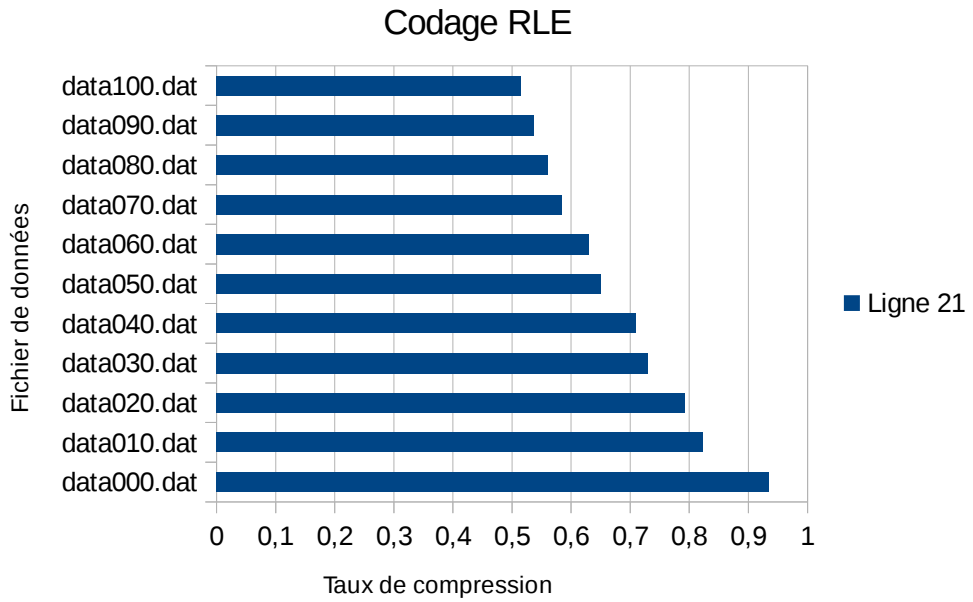
Pour le codage RLE la classe « CRLE » a été utilisée. Cette classe est décrite par les fichiers « CRLE.h » et « CRLE.cpp ». Elle implémente évidemment les méthodes virtuelles de la classe mère mais également d'autres méthodes qui lui sont propres. Voir diagramme de la partie A.

2. Compression / décompression

Notre implémentation établi la liste des symboles grâce à une première lecture des données. Cependant le nombre de symboles (define NB_SYMBOLES_RLE dans CRLE.h) doit être renseigné à l'avance pour permettre le bon déroulement du codage. Le codage s'effectue en suivant la description qui en est faite dans le sujet de tp.

3. Mesure de compression

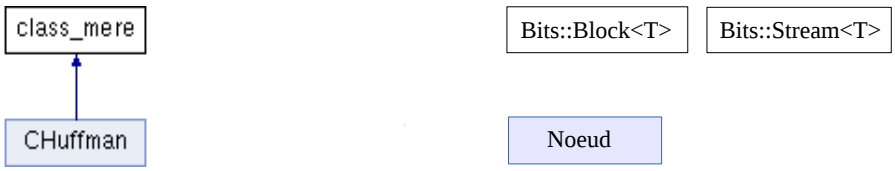
Les mesures de compression ont été effectuées uniquement à partir du nombre de bits résultants du codage des données. Pour que le décodeur puisse effectuer le décodage seul il faudrait en plus des données codées lui transmettre le symbole le moins fréquent, soit 8 bits, ce dernier étant le symbole utilisé pour coder les runs.



D. Codage de Huffman

1. Structure logicielle

Pour le codage de Huffman la classe « CHuffman » a été utilisée. Cette classe est décrite par les fichiers « CHuffman.h » et « CHuffman.cpp ». Elle implémente évidemment les méthodes virtuelles de la classe mère mais également d'autres méthodes qui lui sont propres. Pour établir l'arbre de Huffman nécessaire au codage, la classe « CHuffman » a recours à la classe « Noeud » décrite par les fichiers « Noeud.h » et Noeud.cpp ».

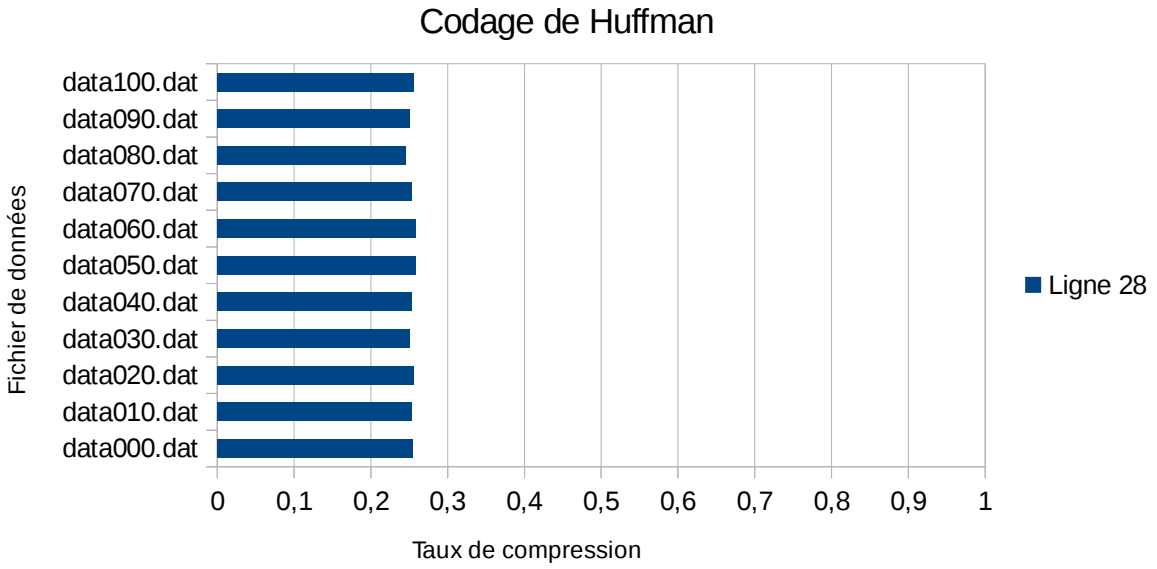


2. Compression / décompression

Notre implémentation établi la liste des symboles grâce à une première lecture des données. Cependant le nombre de symboles (define NB_SYMBOLES_HUFF dans CHuffman.h) doit être renseigné à l'avance pour permettre le bon déroulement du codage. Le codage s'effectue en suivant la description qui en est faite dans le cours. Seul l'arbre de huffman établi par le codeur doit être connu par le décodeur pour que celui-ci effectue le décodage des données.

3. Mesure de compression

Les mesures de compression ont été effectuées uniquement à partir du nombre de bits résultants du codage des données. Pour que le décodeur puisse effectuer le décodage seul il faudrait alors lui transmettre en plus des données codées, une entête contenant la table des symboles ainsi que le code de chaque symbole et le nombre de bits utilisé pour chaque code. De cette manière le décodeur serait en mesure de reconstruire facilement l'arbre de huffman correspondant. Pour un alphabet de 8 symboles comme dans notre cas cela reviendrait à environ 150 bits supplémentaires auxquels viendrait si besoin s'ajouter 8 bits pour indiquer le nombre de symboles dans l'alphabet.



E. Codage par dictionnaire LZW

1. Structure logicielle

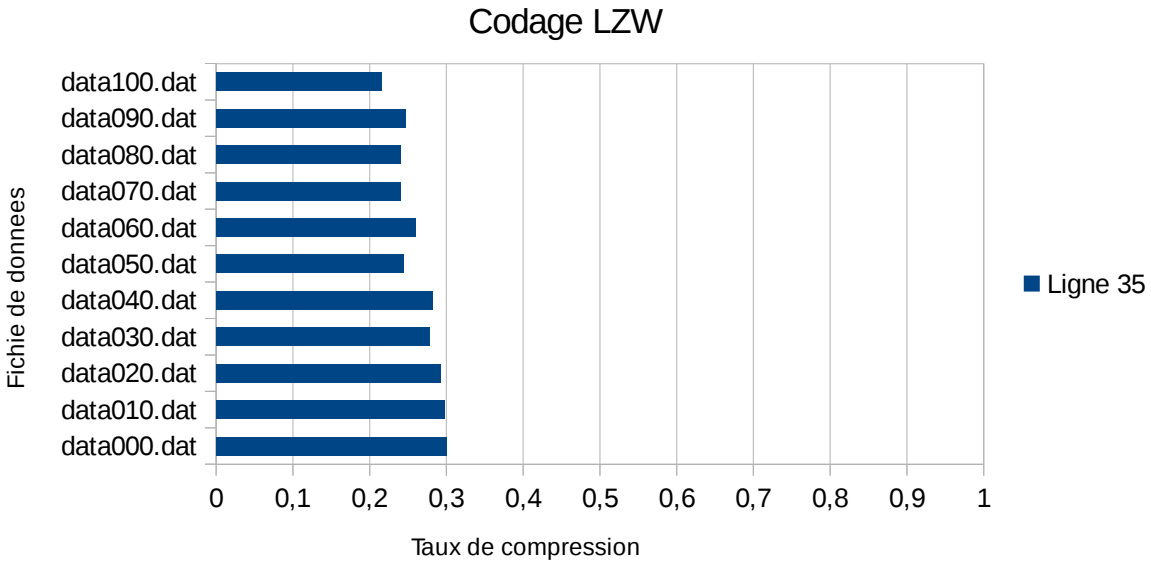
Pour le codage par dictionnaire LZW la classe « LZW » a été utilisée. Cette classe est décrite par les fichiers « LZW.h » et « LZW.cpp ». Elle implémente évidemment les méthodes virtuelles de la classe mère mais également d'autres méthodes qui lui sont propres. Voir diagramme de la partie A.

2. Compression / décompression

Notre implémentation établit la liste des symboles grâce à une première lecture des données. Cependant le nombre de symboles (define NB_SYMBOLES_ALPHABET dans LZW.h) doit être renseigné à l'avance pour permettre le bon déroulement du codage. Le codage s'effectue en suivant la description qui en est faite dans le cours. Le nombre maximum de symboles que peut contenir le dictionnaire doit être défini à l'avance (define NB_SYMBOLES_DICO dans LZW.h).

3. Mesure de compression

Ces mesures ont été effectuées avec un dictionnaires pouvant contenir jusqu'à 255 motifs différents et uniquement à partir du nombre de bits résultants du codage des données. Pour que le décodeur puisse effectuer le décodage seul il faudrait alors lui transmettre en plus des données codées, une entête contenant la table des symboles ainsi que le nombre de symboles dans l'alphabet si besoin, soit au total 72 bits pour un alphabet de 8 symboles.



F. Analyse

On peut remarquer que l'ensemble des codages, à l'exception du codage RLE, offrent des résultats similaires. Cependant des tests sur des fichiers plus grands, avec de nombreuses répétitions de motifs ou au contraire peu de répétitions, et avec un nombre de symboles différents plus ou moins grand, pourraient probablement faire apparaître des différences entre ces différents codages. On remarquera tout de même que le codage RLE qui est le seul à envoyer des caractères non codé offre des résultats bien moins satisfaisant que les autres codages. Un tel codage est en contrepartie certainement un des plus efficace lorsque de nombreuses suites de caractères identiques sont présentes dans les données à coder.