Rapport du projet de compresseur Huffman

HLIN303 – Systèmes d'exploitation
Université de Montpellier

Mattéo Delabre

14 décembre 2016

Dans le cadre de l'unité d'enseignement HLIN303 de L2 informatique, j'ai été amené à développer un programme permettant de compresser des fichiers sans perte, c'est-à-dire de réduire la taille qu'ils occupent tout en conservant l'intégralité des données originales. Je n'ai pas réalisé l'archiveur Python demandé dans le sujet par manque de temps.

Ce programme utilise l'algorithme de compression dit de Huffman, basé sur le codage mis au point en 1952 par David Albert Huffman et toujours utilisé aujourd'hui. Il permet la compression de n'importe quel fichier et sa décompression, en affichant éventuellement des informations détaillées sur la compression.

Dans ce rapport, je décrirai les différents choix techniques que j'ai été amenés à faire lors de la conception de ce programme et les spécifications de l'exécutable final.

1 Choix techniques d'implantation de l'algorithme

1.1 Organisation du code

J'ai divisé le code en plusieurs fichiers gérant différentes parties plus ou moins indépendantes de l'algorithme. La liaison de ces différentes parties est faite dans le programme principal situé dans `main.c`.

`huftree.h` gère la représentation et la manipulation des arbres binaires utilisés dans l'algorithme de Huffman. Il définit notamment une structure récursive `HufVertex` représentant un nœud de l'arbre, son caractère associé, l'effectif de ce caractère dans la source et les sous-arbres gauche et droite. Il permet la construction, l'écriture, la lecture, l'étiquetage et la libération des arbres.

`buffer.h` définit une structure *tampon* permettant d'abstraire l'écriture et la lecture dans un fichier bit par bit, au lieu d'octet par octet. Ceci permet de libérer l'algorithme principal des considérations techniques du bit à bit qui n'est pas supporté nativement.

`compress.h` et `decompress.h` contiennent respectivement les fonctions utiles uniquement aux fonctions de compression et de décompression.

`display.h` définit les diverses fonctions d'affichage.

`common.h` définit les constantes et symboles communs à tout le programme.

`arguments.h` permet l'analyse des arguments passés au programme, en utilisant la bibliothèque *argp* de GNU.

1.2 Format de l'en-tête

L'en-tête du fichier, placé avant les données binaires brutes calculées avec l'algorithme de Huffman, contient des informations sur la longueur originale du fichier et sur l'arbre binaire utilisé pour coder le fichier.

Le stockage de la longueur originale se fait par un entier en 64 bits. Elle permet d'ignorer les bits de remplissage ajoutés à la fin du fichier compressé pour que le nombre de bits soit multiple d'un octet. Autrement, des caractères supplémentaires seraient insérés à la fin du fichier par le processus de décompression.

Le stockage de l'arbre permet le décodage des données binaires. Sachant qu'un 0 est écrit lorsqu'il est nécessaire d'utiliser le sous-arbre gauche pour accéder au caractère et un 1 lorsqu'il faut utiliser le sous-arbre droit, on peut utiliser le procédé inverse pour décoder. On déplace un pointeur partant de la racine à gauche quand on lit un 0 et à droite quand on lit un 1. Si ce pointeur atteint une feuille, il s'agit du caractère décodé.

Pour stocker l'arbre, il faut le linéariser. Pour chaque nœud de l'arbre, le programme écrit un 0 s'il s'agit d'une feuille et 1 sinon. S'il s'agit d'une feuille, le caractère associé à celle-ci suit sur 8 bits. Sinon, on répète récursivement ce procédé pour le sous-arbre gauche et le sous-arbre droit (voir les fonctions `writeTree` et `readTree` dans `huftree.c`).

1.3 Tampon d'écriture bit à bit

Les structures `ReadBuffer` et `WriteBuffer` (voir `buffer.c`) permettent respectivement d'extraire des données d'un fichier bit par bit ou de les écrire également bit par bit. Elles sont notamment utilisées dans les portions du code écrivant ou lisant un arbre vers ou depuis un fichier, ou celles de compression et de décompression.

En interne, elles utilisent un tampon d'un octet et les opérateurs binaires de décalage et de masque. Elles permettent d'alléger les algorithmes et de réduire la fonction de décompression à 35 lignes, commentaires non-compris.

2 Spécifications du programme final

L'exécutable du programme final s'obtient en lançant `make` dans la racine du projet. L'exécutable résultant est `out/huffman`. Il offre l'interface en ligne de commande suivante :

```
# Affiche le message d'aide
huffman --help

# Compresse silencieusement ~/texte.txt vers la sortie standard
huffman ~/texte.txt

# Compresse silencieusement ~/texte.txt vers ~/texte.txt.huf
huffman ~/texte.txt ~/texte.txt.huf
```

```
# Compresse ~/texte.txt vers ~/texte.txt.huf
# en affichant des informations de débogage
huffman -v ~/texte.txt ~/texte.txt.huf

# Décompresse ~/texte.txt.huf vers la sortie standard
huffman -d ~/texte.txt.huf
```

Le programme de décompression présente le désavantage d'avoir un comportement imprévisible s'il est utilisé avec des fichiers qui n'ont pas été compressés avec le programme de compression. Ce problème pourrait être réduit par l'utilisation d'une signature (des octets reconnaissables au début du fichier compressé), mais j'ai choisi de ne pas en utiliser pour garder un fichier compressé léger.