Script de la démonstration

Nicolas CATONI
Alice DEVILDER
Groupe MN03

Pour faire compresser le fichier exemple_huff.txt on utilise les commandes suivantes :

```
adevilde@havok:~/1A/PIM/Projet_2020_2021/src$ adamake compresser.adb
gnatbind -x compresser.ali
gnatlink compresser.ali -g
adevilde@havok:~/1A/PIM/Projet_2020_2021/src$ ./compresser exemple_huff.txt
```

Pour faire décompresser le fichier exemple_huff.txt.hff on utilise les commandes suivantes :

```
ncatoni@n7-ens-lnx055:~$ ./decompresser fichier
```

Avec l'option barvard on obtient :

```
ncatoni@n7-ens-lnx055:~/Documents/PIM/Projet/src/rendu$ ./compresser -b ../exemple_huff.txt
Arbre de Huffman :

(42)
 \--0--(17)
 |     \--0--(8)
 |     |     \--0--(4) 'x'
 |     |     \--1--(4) 'm'
 |     \--1--(9)
 |           \--0--(4)
 |           |     \--0--(2) 'l'
 |           |     \--1--(2)
 |           |           \--0--(1) ':'
 |           |           \--1--(1)
 |           |                 \--0--(0) '~'
 |           |                 \--1--(1) 'd'
 |           \--1--(5) 't'
 \--1--(25)
       \--0--(10)
       |     \--0--(5) ' '
       |     \--1--(5)
       |           \--0--(2) '
'
       |           \--1--(3) 'p'
       \--1--(15) 'e'


Tableau d'encodage des caractères :

'/$' --> 010110
'
 ' --> 1010
'!' --> 100
';' --> 01010
'e' --> 010111
'f' --> 11
'm' --> 0100
'n' --> 001
'q' --> 1011
'u' --> 011
'y' --> 000
```

Architecture en modules

Ces deux programmes utilisent plusieurs modules.

Nous avons tout d'abord implémenté un module *memoire* dans lequel sont définis les types T_byte, T_bit et T_bits (tableau de taille 8 de T_bit) que l'on utilisera tout au long de l'application.

Le module arbre permet d'utiliser les types liés aux arbres et d'effectuer des opérations dessus.

Afin de rendre réutilisable nos sous-programmes associés à la compression d'un fichier, nous avons implémenté un module *code* comprenant les principaux éléments liés à la

construction de l'arbre et de la table de codage de Huffman mais également les procédures de réécriture du fichier à l'aide du codage de Huffman.
De même, le module *decode* contient les éléments nécessaires à la décompression : le programme va appeler des sous-programmes de *decode* afin de reconstruire l'arbre de Huffman suivant un parcours infixe et pouvoir décoder le fichier compressé.

Principaux choix réalisés

Pour créer l'arbre de huffman, il faut d'abord créer un tableau de chaque feuille. Ensuite, on extrait le nœud à fréquence minimale du tableau, pour le fusionner avec de deuxième nœud minimum, jusqu'à ce que la fréquence du nœud minimum soit égale au nombre de caractères dans le texte.

Pour gérer la construction de l'arbre, on utilise une liste d'arbres de taille 256. La construction de l'arbre se fait en respectant plusieurs conditions notamment que la fréquence du fils gauche d'un arbre est plus faible que celle du fils droit. De plus, la fréquence d'un nœud correspond à la somme des fréquences de ses fils droit et gauche. Ainsi, nous avons décidé de chercher les 2 arbres de la liste avec les plus faibles fréquences. Puis l'arbre de Huffman est construit itérativement.

Pour stocker le code de huffman de chaque symbole, on utilise un tableau d'enregistrement d'une valeur (tableau de 256 bits) et d'une longueur qui nous permet d'obtenir le code de Huffman. Par exemple, si la longueur du ième tableau vaut n, alors les n premiers bits de la valeur correspondent au code de Huffman du ième caractère de la table ASCII.

Pour vérifier si un caractère est une fin de fichier, nous avons introduit une variable "ff" (Booléen) dans le type T_noeud qui est vrai quand c'est une fin de fichier et faux sinon.

Pour décompresser le fichier on commence par identifier les segments du fichier compresser en déterminant la taille du tableau des bytes dans l'ordre du parcours infixe puis la taille du parcours infixe. On lit ensuite ces deux premiers segments ce qui nous permet de reconstruire l'arbre de huffman. On lit ensuite le segment message du fichier compressé.