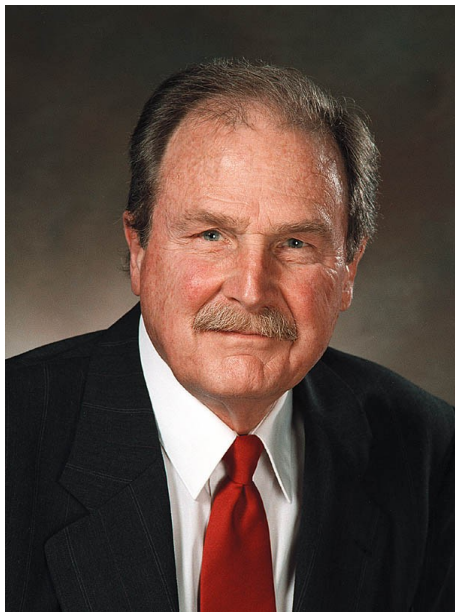
Ocaml - Projet 2014

Compression de Huffman



Eric UZENAT, Mohamed YENNEK

Table des matières

1. Mode d'emploi :....3
1.1 Présentation :....3
1.2 Compilation :....3
1.3 Exécution :....3
1.4 Création de la documentation....3
2. Implémentation du programme :....3
2.1 Diagramme des modules :....3
2.2 Description :....4
2.2.1 Histogramme :....4
2.2.2 Treehuff :....4
2.2.3 Tas min :....4
2.2.4 Lexique :....4
2.2.5 Bitio :....5
2.2.6 Compression :....5
2.2.7 Décompression :....5
2.2.8 Main :....5
3. Bugs rencontrés :....5
Conclusion :....6

1. Mode d'emploi :

1.1 Présentation :

Le projet a pour objectif de pouvoir compresser et décompresser un fichier texte grâce au très célèbre algorithme de Huffman.

1.2 Compilation :

Pour compiler ou recompiler le programme, un *makefile* est fourni avec le code source. Pour compiler le projet, il suffira de lancer la commande : « `make huffman` » Il est aussi possible de supprimer tous les fichiers créés lors de la compilation en rajoutant *clean* : « `make clean` ».

1.3 Exécution :

Pour lancer le programme, il suffit de taper la commande suivante:

« `./huffman [-c ou -d]` *`comp.hf fichier`* »

Le programme doit être lancé obligatoirement avec deux options possibles :

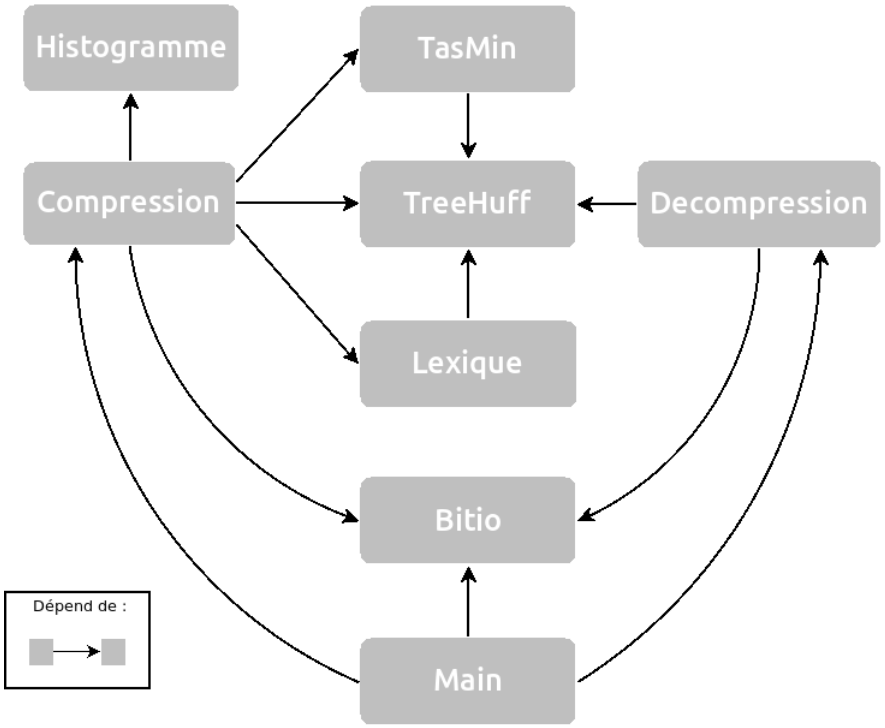
- `-c` qui correspond à l'option de compression :
  « *`fichier`* » est compressé vers « *`comp.hf`* »
- `-d` qui correspond à l'option de décompression :
  « *`comp.hf`* » est decompressé vers « *`fichier`* »

1.4 Création de la documentation

Le *makefile* offre aussi la possibilité de générer automatiquement la documentation en html. En tapant la commande « `make doc` », un fichier doc.html sera créé dans le répertoire courant où il sera alors directement possible d'accéder à la documentation du programme.

2. Implémentation du programme :

2.1 Diagramme des modules :



2.2 Description :

2.2.1 Histogramme :

L'histogramme est une structure d'arbre binaire de recherche qui contient un tuple avec un caractère et sa fréquence (l'ABR est construit à partir des caractères contenus dans le fichier à compresser). A chaque fois que l'on rencontre un caractère, on l'ajoute dans l'arbre si ce dernier ne le contenait pas déjà ; dans le cas contraire on l'incrémente de 1.

2.2.2 Treehuff :

Le module Treehuff contient l'arbre de Huffman ainsi que quelques méthodes associées.

2.2.3 Tas min :

Une fois l'histogramme créé, on construit un tas min. Initialement, on a plusieurs tas min où chaque tas min représente une feuille de treehuff. Ensuite, on prend à chaque fois les deux tas min de poids minimum puis on les fusionne et on les réinjecte, et tout cela en un

temps optimal.

2.2.4 Lexique :

Le lexique est un ABR qui contient des tuples avec comme clés des caractères et comme valeurs une liste de leur codage en binaire qui à été obtenue à partir de l'arbre treehuff.

2.2.5 Bitio :

Une bibliothèque qui permet de manipuler les fichiers Unix comme si c'était des flots de bits.

2.2.6 Compression :

- On écrit les quatre octets magiques ayant la valeur 87_{16}, $4A_{16}$, $1F_{16}$, 48_{16} .
- On écrit un octet de valeur `n` suivi de `n` octets ignorés par le décompresseur.
- On écrit la représentation de l'arbre.
- On écrit la suite de codes compressés.
- On écrit un octet `m` suivi de `m` octets ignorés.
- On écrit le code de fin de fichier.

2.2.7 Décompression :

- On lit les quatre premiers octets, on teste que ce sont bien les quatre octets magiques.
- On lit l'octet suivant de valeur n et on déplace la tête de lecture de n octets vers l'avant.
- On lit la suite d'octets suivants afin de reconstruire l'arbre de Huffman qui a servi à la compression.
- On lit bit par bit le code compressé et en utilisant l'arbre on reconstitue le texte d'origine.

2.2.8 Main :

Le main se charge d'appeler les modules Compression et Décompression.

3. Bugs rencontrés :

On a rencontré quelques bugs après de nombreux tests qui sont :

- impossible de compresser les fichier de plus de `1Mo`, le problème est un `StackOverflow`, donc afin de le résoudre on a tout codé en récursion terminale et donc le problème est résolu.

- Impossible de compresser un fichier audio ou image car notre programme rencontre des caractères de fin de fichier (*\003*) dans le fichier avant la fin. Donc, il y a une grande perte de données. Afin de résoudre ce problème, nous avons changé le symbole marquant la fin du codage, il correspond maintenant à 256 (il fallait utiliser un entier autre que ceux utilisés dans la table ascii) donc le problème est résolu.

Conclusion :

Ce projet nous a permis de mieux saisir un nouveau langage et surtout une nouvelle manière de programmer. Mais aussi d'apprendre un algorithme permettant la compression des données sans perte.