Compression selon Huffman

Juliusz Chroboczek

25 février 2011

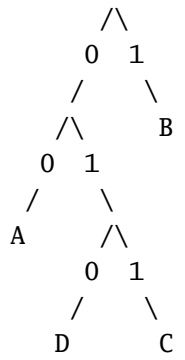
Introduction

Le but de ce projet est d'écrire en Haskell un compresseur et un décompresseur, semblables à *gzip*, mais utilisant l'algorithme de Huffman.

1 Codes préfixes et arbres binaires

L'idée fondamentale de l'algorithme de Huffman est de coder les valeurs possibles des octets du fichier d'entrée (les « symboles ») par un nombre variable de bits : les symboles les plus courants seront codés par un petit nombre de bits, tandis que les plus rares par un nombre de bits plus grand.

Le codage que construit l'algorithme de Huffman est un *codage préfixe* : aucun code n'est un préfixe d'un autre. Un codage préfixe peut être représenté par un arbre binaire, où les symboles à coder sont placés aux feuilles : le code associé à un symboles est obtenu en suivant le chemin menant à ce symbole depuis la racine de l'arbre, où un parcours vers la gauche représente le bit 0, et un parcours vers la droite le bit 1.



Symbole	Code
A	00
B	1
C	011
D	010

2 Algorithme de compression

L'algorithme de compression a trois étapes : la construction d'un histogramme, la construction de l'arbre binaire, et la compression proprement dite.

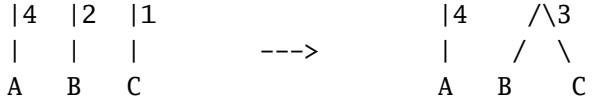
2.1 Construction de l'histogramme

La première étape consiste à construire un histogramme, c'est à dire une table qui à chaque symbole associe le nombre de fois que celui-ci apparaît dans le fichier à compresser (sa « fréquence »).

2.2 Construction de l'arbre

Étant donné un arbre Λ, on définit son *poids* comme suit :
- si Λ est une feuille, alors son poids est la fréquence du symbole correspondant ;
- sinon, le poids de Λ est la somme des poids de ses deux fils.

La construction de l'arbre (l'algorithme de Huffman proprement dit) est un algorithme itératif opérant sur une forêt (une collection d'arbres) triée par poids [1]. On commence avec la forêt triviale consistant d'une collection de feuilles, une par symbole. À chaque étape de l'algorithme, on prend les deux arbres de moindre poids (les deux derniers éléments de la forêt), qu'on enlève de la forêt et qu'on combine en un seul arbre. On insère ensuite l'arbre ainsi obtenu au bon endroit dans la forêt.



L'algorithme termine lorsque la forêt ne contient plus qu'un seul arbre.

Le symbole de fin de fichier Nous aurons besoin d'un symbole supplémentaire qui indique la fin du fichier compressé. Avant de construire l'arbre de Huffman, vous devrez insérer dans la forêt une feuille supplémentaire, de poids 1, représentant le symbole de fin de fichier.

2.3 Stockage des données binaires

Dans les sctions suivantes, vous aurez besoin de stocker des flots de bits sur disque ; or les systèmes Unix et Windows permettent de stocker des flots d'octets.

Un flot de bits sera stocké sur disque en ordre *gros-boutiste*. Par exemple, la suite de bits

0111 0010 0110 1111 0111 0010 0110 1111

sera stockée comme la suite d'octets 72_{16}, $6F_{16}$, 72_{16}, $6F_{16}$.

2.4 Stockage de l'arbre

Afin de pouvoir décompresser un fichier compressé à l'aide de l'algorithme de Huffman, il faut connaître l'arbre utilisé. Vous devrez donc le stocker dans le fichier compressé.

Pour cela, parcourez l'arbre en profondeur. À chaque fois que vous rencontrez un nœud interne, stockez un bit 1 suivi de la représentation du fils gauche suivie immédiatement de la représentation du fils droit. Lorsque vous rencontrez une feuille portant le symbole s, écrivez un bit 0 suivi
- des 8 bits de s, en ordre gros-boutiste, si $s \leq 254$;

1 Dans quel sens ?

– des 9 bits 1111 1111 0 si $s = 255$;
– des 9 bits 1111 1111 1 si s est le symbole de fin de fichier.

2.5 Compression

La compression consiste simplement à lire le fichier et, pour chaque octet lu, écrire le symbole correspondant. À la fin du fichier, il faudra ajouter le code du symbole de fin de fichier.

Pour cela, vous aurez besoin de construire une table qui à chaque symbole associe le code correspondant. À vous de voir si vous voulez inclure le symbole de fin de fichier dans cette table, ou si vous préferez le traiter séparément.

3 Décompression

Pour décompresser un fichier, il faut commencer par lire l'arbre binaire stocké dans celui-ci. Une fois l'arbre reconstruit, on lit le fichier bit-à-bit tout en parcourant l'arbre ; à chaque fois qu'on arrive à une feuille, on écrit le symbole associé dans le fichier de sortie, et on revient à la racine.

Le processus se termine lorsqu'on rencontre le symbole de fin de fichier. Si le fichier d'entrée se termine sans qu'on ait rencontré celui-ci, il faut indiquer une erreur.

4 Format de fichier

Votre compresseur/décompresseur devra être *interopérable* avec celui que j'ai écrit — un fichier généré par votre code devra pouvoir être interprété par le mien, et un fichier généré par le mien devra pouvoir être interprété par le vôtre.

Un fichier compressé aura l'extension « `.hf` » et sera structuré comme suit :
– 4 octets « magiques » ayant la valeur 87_{16}, $4A_{16}$, $1F_{16}$, 48_{16} ;
– un octet de valeur n suivi de n octets ; ces $n + 1$ octets seront ignorés par le décompresseur ;
– la représentation de l'arbre ;
– la suite de symboles compressés ;
– 0 à 7 bits valant 0, de façon à aligner la suite du fichier sur une frontière d'octet ;
– un octet m suivi de m octets ignorés.

5 Extensions

Toutes les extensions seront les bienvenues, et leur présence sera prise en compte lors de l'évaluation.

Les deux plages de n et m octets décrites ci-dessus sont réservées à des extensions au format de fichier. Si vous voulez vous en servir, choisissez un nombre magique de quatre octets globalement unique, par exemple en le tirant au hasard [2], et stockez dans les plages réservées votre nombre magique suivi des données dont vous avez besoin. À la lecture, vous devrez ignorer les zones réservées si elles ne commencent pas par votre nombre magique.

2 Je vous l'avais bien dit que c'était magique.

6 Évaluation

Nous nous réservons le droit d'évaluer le projet selon n'importe quels critères, y inclus les suivants :

- le fait que votre programme fonctionne ou pas ;
- le fait que votre programme interopère avec le mien ;
- le fait que votre programme se comporte comme un utilitaire Unix ;
- la nature fonctionnelle ou impérative des structures de données utilisées ;
- la modularité de votre programme (par exemple, le fait qu'une bibliothèque d'entrées/sorties bit-à-bit soit identifiable) ;
- la vitesse d'exécution [3] et l'utilisation de mémoire de votre programme ;
- le sentiment général de qualité qui émane de votre programme.

7 Modalités de soumission

Votre projet devra être soumis sous forme d'une archive *tar* compressée, soit à l'aide de votre programme [4], soit à l'aide de *gzip*. Elle sera envoyée par courrier électronique à l'adresse `jch@pps.jussieu.fr` avant une date qui sera précisée ultérieurement sur la page *web* du cours. Le courrier électronique que vous m'enverrez devra *impérativement* avoir le sujet « `Projet PFA` ».

L'archive devra s'appeler *prénom–nom*`.tar.gz` ou *prénom–nom*`.tar.hf`, où *prénom* est votre prénom et *nom* est votre nom, et s'extraire dans un sous-répertoire *prénom–nom* du répertoire courant. Par exemple, si vous vous appelez *Louis Pasteur*, votre archive devra porter le nom `louis-pasteur.tar.gz` ou `louis-pasteur.tar.hf`, et son extraction devra créer un répertoire `louis-pasteur` contenant tous les fichiers que vous nous soumettrez.

L'archive que vous soumettrez devra contenir :

- les sources *en Haskell* de votre compresseur/décompresseur ;
- un fichier *README* me donnant toutes informations supplémentaires qui peuvent m'intéresser.

Vous avez clairement intérêt à mentionner les points forts de votre solution dans le *README* ; à vous de voir si vous voulez mentionner les points faibles.

3 Ben ouais.

4 Ce qui fera bonne impression, mais risque de vous pénaliser si nous n'arrivons pas à la décompresser.