Mini-projet : codage de Huffman

Laure Danthony et Sylvain Four

1 Algorithmique et programmation

1.1 Codage de Huffman standard

- idée : les occurences les plus fortes d'une lettre sont représentées par des codages plus courts.
- implémentation :
  - codage : construction de l'arbre de codage en parcourant entièrement le texte à coder. Noter qu'on ne compte que les caractères qui sont utilisés dans le texte. Puis construction du fichier compressé en mettant en premier le codage de l'arbre, puis le texte codé lui-même.
  - décodage : reconstruction de l'arbre, puis parcours du texte tout en le décodant. Il n'y a pas d'ambiguité dans le décodage des caractères, lorsque l'on arrive à une feuille de l'arbre on est certain d'avoir décodé le bon caractère (propriété de l'algorithme de Huffman).
- spécifications :
  - codage de l'arbre : on définit le codage d'un arbre récursivement. Le codage d'un noeud est 1 auquel on concatène le codage de son fils gauche puis le codage de son fils droit. Le codage d'une feuille est 0 auquel on concatène le codage sur 8 bits du caractère associé.
  - lecture et écriture dans des fichiers : elle ne peut se faire que 8 bites à la fois ... On est donc obligé de remplir le fichier .zip avec un nombre de bites multiples de 8. L'astuce consiste à ajouter les $0 \leq n < 8$ bits supplémentaires qui seront des 0 inutiles. Pour que le codage reste sans ambiguité, on code ce nombre n au début du fichier .zip. Cela évitera de lire ces zéros inutiles.
  - gestion de mémoire : on bufferise les octets lus dans le fichier. Pour la compression et la décompression on a choisi une taille de buffer permettant de limiter au plus la complexité de notre implémentation. Pour cela, on a été contraint de faire de multiples tests.

1.2 Codage de Huffman amélioré

1.2.1 Première approche

Cette approche consiste à passer en entête une liste d'arbres : on passe un arbre pour chaque premier caractère d'un digramme présent dans le texte initial.

On se donne l'ensemble des digrammes présents dans le texte commençant par le caractère c_1 puis on lui crée un arbre associé par la méthode traditionnel excepté qu'on place le caractère c_2 dans l'arbre non pas par rapport à sa fréquence propre mais par rapport à la fréquence de c_1c_2 dans le texte.

Pour coder le texte, on commence par coder "en dur" son premier caractère c_1 (c'est-à-dire sur 8 bits) puis on code le caractère suivant c_2 en prenant dans l'arbre associé à c_1 le codage associé à c_2. Puis pour coder le caractère c_3, on prend son code dans l'arbre associé au caractère c_2...

Le décodage s'en déduit aisément une fois qu'on a la liste des arbres.

On code cette liste d'arbres de 2 manières possibles :

- soit on code le tableau entier de 256 arbres. Pour cela, pour i allant de 1 à 255, s'il existe un arbre associé au caractère de code ASCII i on écrit 1 puis on écrit le codage de cette arbre par la méthode décrite précédemment ;
- soit on code seulement les arbres réellement existant. Pour cela, on écrit d'abord le nombre d'arbres que l'on va coder puis on code sur 8 bits chaque caractère concerné suivi du codage de son arbre.

On décide de choisir l'une ou l'autre de ces 2 méthodes suivant que le nombre d'arbres est supérieur ou non à 31.

1.2.2 Deuxième approche

Cette approche est très proche de la précédente dans la mesure où on code aussi une liste d'arbres. Mais, cette fois-ci on code un arbre par paire de caractères (c_1, c_2) tel qu'il existe un caractère u tel que le trigramme c_1c_2u apparaisse.

Pour coder le texte, on commence par coder "en dur" ses deux premiers caractères c_1 et c_2 (c'est-à-dire sur 8 bits chacun) puis on code le caractère suivant c_3 en prenant dans l'arbre associé à c_1 et c_2 le codage associé à c_3. Puis pour coder le caractère c_4, on prend son code dans l'arbre associé aux caractères c_2 et c_3...

Pour coder la liste des arbres on prend la même méthode excepté le fait qu'ici le critère change. Le critère est que le nombre d'arbres est supérieur ou non à 4095.

2 Programme général

Notre programme général fait le choix entre les 4 méthodes suivantes (sans oublier qu'il refuse de compresser si le fichier de départ est de taille insuffisante) :

- la méthode 0. Si le fichier est de trop petite taille (c'est-à-dire si la méhode 1 crée un fichier compressé plus grand que celui d'origine) le fichier compressé n'est autre que le fichier initial auquel on ajoute en entrée les 2 bits 00 pour noter qu'il est compressé ainsi ;
- la méthode 1. C'est la méthode du codage de Huffman traditionnel. On opte pour celle-ci lorsque la méthode 2 construit un fichier de taille supérieure à celle du fichier compressé par la méthode 1. Le fichier compressé est ici précédé par les 2 bits 01 ;
- la méthode 2. C'est la méthode qui consiste à passer en entête un arbre par premier caractère de digrammes apparaissant dans le fichier. On choisit

cette méthode lorsque la méthode 3 construit un fichier de taille supérieure à celle du fichier compressé par la méthode 2. Le fichier compressé est ici précédé par les 2 bits 10 ;

– la méthode 3. C'est la méthode qui consiste à passer en entête un arbre par couples de caractères débutant les trigrammes apparaissant dans le fichier. On choisit cette méthode les autres fois. Le fichier compressé est ici précédé par les 2 bits 11.

3 Pour faire tourner le programme

3.1 Compilation avec ocamlopt

L'utilisateur de notre `zip/dezip` doit au prélable compiler les sources ocaml de la façon suivante :

- Pour l'exécutable `zip` : `ocamlopt -o zip zip.ml`
- Pour l'exécutable `dezip` : `ocamlopt -o dezip dezip.ml`

Les fichiers créés sont des exécutables. Remarquons l'utilisation de `ocamlopt` à la place de `ocamlc` (cela permet une plus grande vitesse d'exécution).

3.2 Exécution des programmes

- Pour compresser un fichier "`file`", il suffit de taper la ligne de commande : `./zip file`.
- Pour décompresser un fichier "`file.zipn`", il suffit de taper la ligne de commande : `./dezip file.zipn.` UNotons que il n'y a pas destruction de `file.zipn` (choix des programmeurs).