Codage de Huffman

Jean-Baptiste Rouquier

1er janvier 2005

1 Aspects pratiques

Ce TP réutilise le TP précédent sur les tas (TP «Heapsort»). Pour utiliser le corrigé[1] :

- recopiez les fichiers dynarray.ml, dynarray.mli, heap.ml et heap.mli dans un répertoire de votre home,
- compilez-les par la commande
  `ocamlc -c dynarray.mli dynarray.ml heap.mli heap.ml`
- ajoutez les lignes suivantes en tête de votre fichier `login.ml`
  ```
  #load "dynarray.cmo"
  #load "heap.cmo"
  ```
- lisez le fichier heap.mli pour connaître les fonctions disponibles[2].

Si un essai de codage et décodage d'un message avec l'arbre optimal marche (question 3.3), il n'est pas nécessaire de tester les fonctions intermédiaires.

2 Codage et décodage

Soit m un message à transmettre composé de lettres de l'alphabet A. Le principe du codage de Huffman est d'utiliser un arbre binaire déséquilibré dont les feuilles sont les lettres de A. Au lieu d'écrire le rang de la lettre dans l'alphabet (ce qui prend $\log |A|$ bits), on écrit le chemin menant de la racine à cette lettre (ce qui prend autant de bits que la longueur du chemin). Ainsi, si les lettres fréquentes sont à une faible profondeur, moins de bits seront nécessaires.

Le message codé est la concaténation des codes des lettres du message en clair.

Question 2.1 (bonus). Montrer que l'on a bien défini une fonction injective.

On utilisera pour alphabet l'ensemble des 256 caractères ASCII (voir le module `Char` de la bibliothèque standard), mais cet algorithme s'applique à n'importe quel alphabet.

Pour représenter un arbre de Huffman, on utilisera le type suivant :

```
type huffman =
  | Noeud of huffman * huffman
  | Feuille of char
```

Le chemin de la racine à une feuille sera représenté par une `direction list` où

```
type direction = Gauche | Droite
```

Question 2.2. Écrire la fonction `dictionnaire : huffman -> direction list array` qui renvoie un tableau de taille 256 dont la case i contient le codage (c'est-à-dire le chemin depuis la racine) du caractère `Char.chr i`.

Remarque. Si vous trouvez cela plus commode pour cette question et la suivante, vous pouvez renvoyer le miroir (la liste retournée) du codage au lieu du codage lui-même.

[1] Il a été légèrement modifié pour permettre une fonction de comparaison arbitraire dans les tas.

[2] ou bien exécutez
`mkdir doc ; ocamldoc -d doc -html -colorize-code dynarray.mli heap.mli`
pour en avoir une jolie version html.

Question 2.3. Écrire la fonction `zip : huffman -> string -> direction list` qui encode le message.

Question 2.4. Écrire `decode_un_char : direction list -> huffman -> direction list * char` qui décode un caractère et renvoie la suite du message codé.

Question 2.5. Écrire `unzip : huffman -> direction list -> string` qui décode le message.

3 Compression

Le but est maintenant de calculer l'arbre de Huffman qui donnera le message codé le plus court.

Question 3.1. Montrer qu'il existe un arbre optimal dans lequel les deux lettres a et b les moins fréquentes sont sur des feuilles sœurs. Indication : considérer un arbre optimal et montrer que l'on peut le modifier en un autre arbre optimal vérifiant cette propriété.

On considère maintenant les lettres comme des feuilles, et l'on remplace les deux lettres les moins fréquentes a et b par un arbre ayant a et b pour feuilles et une racine. On attribue la fréquence somme des fréquences de a et b à cet arbre. Soit ℓ la liste contenant ces feuilles et cet arbre.

La question précédente a montré qu'il existe un arbre optimal tel que tous les arbres de la liste ℓ en soient des sous-arbres. On peut refaire le raisnonnement précédent : il existe un arbre optimal où les deux arbres c et d les moins fréquents de ℓ sont frères. On peut donc remplacer c et d par un arbre dont la racine a c et d pour fils, et dont la fréquence est la somme des fréquences de c et d. On continue ainsi jusqu'à ce que ℓ ne contienne plus qu'un arbre, c'est l'arbre de Huffman cherché.

Question 3.2. Écrire `best_huffman : string -> huffman` qui calcule l'arbre de Huffman optimal pour le texte donné. C'est ici que l'on réutilise les tas pour représenter ℓ efficacement (on pourra écrire une première version naïve utilisant une liste).

Question 3.3. Faire un test de codage et décodage.

Remarque. Cet algorithme est optimal si chaque caractère est indépendant des autres : on considère le message à coder comme une suite aléatoire de lettres, chaque lettre ayant la même probabilité d'apparaître à toutes les positions, mais avec des probabilités différentes entre les lettres (donc des fréquences différentes dans le message).

Ceci n'est pas le cas d'un message en langue naturelle (ni même de la plupart des fichiers informatiques non compressés). En français par exemple, il y a une très forte probabilité d'avoir un «u» après un «q». Il existe d'autres algorithmes tenant compte de cette observation, comme celui utilisé par l'outil de compression gzip.