Programmation CAML 2

Durée 2 heures – Aucun document autorisé

Le but de ce problème est de mettre en œuvre le procédé de codage de Huffman. Les types et les exemples sont prédéfinis dans le fichier huffman.ml

Le codage de Huffman est un procédé de compression des données élaboré en 1952 par David Albert Huffman. Il appartient à la famille des codages statistiques (comme le Morse) qui reposent sur le principe suivant :

Soit un alphabet de 16 caractères différents. Si on utilise un codage de taille constant, il faudra 4 bits pour représenter un caractère. Alors un texte de N caractères nécessitera toujours 4*N bits.

Mais si certains caractères sont très fréquents et d'autres très rares, on peut choisir des codes courts (1 ou 2 bits) pour les caractères fréquents quitte à avoir des codes beaucoup plus longs pour les caractères rares. On peut alors espérer statistiquement réduire la taille du texte codé.

Le codage de Huffman est un code de ce type. On distingue trois étapes :

1. **Construction** : A partir d'un langage défini par son alphabet (l'ensemble des caractères qu'il utilise) et par une fonction donnant la probabilité d'utilisation de ces caractères, nous allons construire un arbre binaire appelé arbre de Huffman, caractéristique du langage.
2. **Codage** : A partir d'un arbre de Hufmann et d'un texte à coder, on cherchera à construire son code de Huffman
3. **Décodage** : A partir d'un arbre de Huffman et d'un texte codé, on cherchera à décoder le texte.

I. Arbre de Huffman

Un langage est caractérisé par la liste des caractères qu'il utilise (son alphabet) et leur probabilité d'utilisation. Nous utiliserons l'exemple suivant :

Caractère	'h'	'e'	'l'	'o'	' '
probabilité	0.1	0.2	0.4	0.2	0.1

Nous représenterons donc en Caml un langage par le type produit suivant :

```
type langage = {alphabet:char list ; fProba:char->float};;
```

et nous utiliserons le langage donné dans l'exemple précédent :

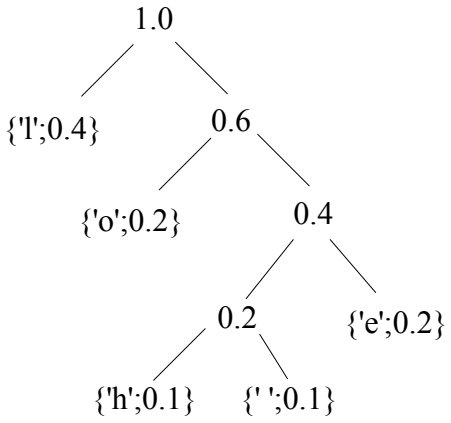
```
let f= function
`l`->0.4
|`h`->0.1
|` `->0.1
|`e`->0.2
|`o`->0.2
|_->0.;;
let alpha=[`h`;` `;`e`;`o`;`l`];;
let lang ={alphabet=alpha;fProba=f};;
```

Le code de Huffman repose sur la construction et l'utilisation d'un arbre de Huffman qui modélise la probabilité d'utilisation des différents caractères. Il est constitué soit d'une feuille (un caractère et sa probabilité) soit d'un nœud constitué d'un arbre gauche, d'un arbre droit et de la somme de leurs probabilités.

On définit un arbre de Huffman par le type somme récursif suivant :

```
type feuille={car : char ; proba : float};;
type arbreH=   Feuille of feuille
             | Noeud of arbreH * float * arbreH;;
```

L'arbre défini dans le fichier huffman.ml peut être représenté par :



Pour chaque fonction, vous devez obligatoirement respecter la forme des exemples d'exécution du fichier huffman.ml.En particulier les types, la nature fonctionnelle et pour les fonctionnelles, l'ordre des paramètres.

- Ecrire une fonction ***probabilité*** qui à un arbre de Huffman associe sa probabilité.
- Ecrire une fonction ***hauteur*** qui à un arbre de Huffman associe sa hauteur (la plus grande distance entre sa racine et une de ses feuilles)
- Ecrire une fonctionnelle ***fusion*** permettant de fusionner deux arbres de Huffman

II. Construction de l'arbre de Huffman

La construction de l'arbre de Huffman à partir d'un langage va suivre les étapes suivantes

- Ecrire une fonction ***listeFeuilles*** qui à partir d'un langage construit la liste des feuilles correspondant à chacun des caractères. On suppose que l'alphabet est trié dans l'ordre croissant des probabilités d'utilisation. La liste des feuilles sera donc également triée dans l'ordre croissant des probabilités.

- Ecrire une fonction ***insère*** qui insère un arbre de Huffman à sa place dans une liste triée d'arbres de Huffman.

- Ecrire une fonction ***build*** qui à partir d'une liste triée d'arbres de huffman, produit récursivement un unique arbre en fusionnant à chaque étape les deux arbres de plus faibles probabilités et en insérant l'arbre ainsi fusionné à sa place pour que la liste reste triée.

- Ecrire enfin la fonction ***construitAH*** qui à partir d'un langage construit son arbre de Huffman.

III. Codage de Huffman

On souhaite dans cette partie, déterminer le code de chaque caractère puis coder une chaîne de caractères donnée.

Le principe est le suivant : Chaque caractère est codé par un mot binaire qui caractérise le chemin entre la racine de l'arbre et la feuille correspondant au caractère : plus précisément, à partir de la racine, on ajoute '0' en tête du code à chaque fois qu'on prend l'arbre gauche et '1' à chaque fois qu'on prend l'arbre droit.
On obtiendra donc avec notre exemple les codes suivants :

Caractère	'h'	'e'	'l'	'o'	' '
Code	"1100"	"111"	"0"	"10"	"1101"

(On remarque que les caractères les plus utilisés sont plus courts).

↳ Ecrire une fonction ***codage*** qui à partir d'un arbre de Huffman construit la liste constituée des couples (caractère, mot binaire).

On pourra écrire d'abord une fonctionnelle récursive ***ajout*** qui ajoute un caractère (0 ou 1) en tête de tous les mots binaires d'une liste de couples (caractère, mot binaire).

↳ Ecrire une fonctionnelle ***codeCar*** qui à un caractère c et une liste de couples (caractère, mot binaire) donne le mot binaire associé au caractère c.

↳ Ecrire une fonction ***codeTexte*** qui donne la séquence binaire codant une chaîne de caractère donnée.

Le code de la chaîne "hello leo" est alors "11001110010110101110" qui code 9 lettres en 21 bits au lieu de 27 pour un code constant.

IV. Décodage

On cherche maintenant à décoder une séquence binaire à l'aide d'un arbre de Huffman.
Le fichier huffman.ml contient toutes les primitives sur les chaînes de caractères et permet donc l'utilisation des fonctions *tetec* et *reste* donnant l'initiale et le reste d'une chaîne de caractères.

Le principe du décodage d'une séquence binaire est le suivant : en partant de la racine de l'arbre de Huffman, on lit les caractères de la chaîne un à un en prenant l'arbre gauche quand on lit un 0 et l'arbre droit quand on lit un 1 jusqu'à arriver sur une feuille. On obtient alors un caractère (celui correspondant à la feuille) et on recommence avec le reste de la séquence binaire jusqu'à ce qu'elle soit vide.

↳ Ecrire une fonctionnelle récursive ***decodeCar*** qui à une séquence binaire et un arbre de Huffman associe le caractère décodé et la séquence restante.

↳ Ecrire enfin une fonctionnelle récursive ***decodeTexte*** qui à un arbre de Huffman et une séquence binaire associe le texte décodé.

La séquence binaire "11001110010110101110" doit être décodée par "hello leo".