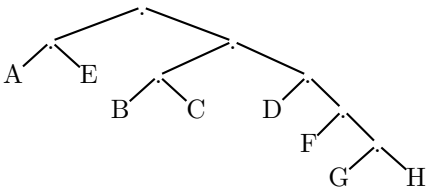
Option Informatique
Arbre de Huffman (ESIM 97)

Sujet

30 mars 2007

Les codes employés classiquement en informatique sont des codes à longueur fixe (par exemple le code ASCII). Ils présentent l'avantage de la simplicité (on n'a aucun problème pour reconnaître la fin du code d'un caractère), mais ils ne sont pas économes en place. Le code de Huffman est un code à longueur variable. Il exploite la répartition statistique des fréquences de caractères : les caractères les plus fréquents sont codés avec «peu de bits», les moins fréquents avec «plus de bits». De plus, il possède une propriété remarquable : aucun code n'est le début d'un autre.

Les caractères à coder se trouvent sur les deuilles d'un arbre binaire et la suite de bits permettant de coder un caractère représente le chemin conduisant de la racine à la feuille le contenant. Voici un exemple d'arbre permettant de codes les huits premières lettres de l'alphabet.



Si on convient de représenter les branches de gauche par des 0 et les branches droites par des 1, la suite de bits codant le H est 11111, car on n'a parcouru que des branches droites pour aller de la racine à la feuille contenant H. De même, la code de A est 00, celui de B es 100, celui de C est 101 ; etc.

Chaque noeud de l'arbre contient comme informaiton une paire constituée d'une chaîne (champs `chaine`) qui contient tous les caractères situés en dessous de lui et d'un entier (champs `poids`) qui est égal à la somme des fréquences de ces caractères. De plus, chaque noeud possède aussi deux fils, un fils gauche (`fg`) et un fils droit (`fd`).

Algorithme de construciton de l'arbre

a On crée la liste (fonction `creer_liste`) d'arbres réduits à un seul noeud. Chaque noeud de cette liste a pour information une paire consitutée d'une lettre et de son poids, les deux fils de chacun de ces noeuds sont initialisés avec `nil`. La liste doit être ordonnée par ordre de poids croissant. Tous les noeuds de cette liste sont en fait des feuilles de l'arbre de Huffman.

   Pour simplifier l'écriture, on ne représentera, ci-dessous seulement, les arbres de la liste que par leur noeud racine sans leurs fils.

   Au départ, la liste peut être ainsi représentée :
   `lst` →`(G ;1)`→`(H ;1)`→`(F ;2)`→`(B ;4)`→`(C ;4)`→`(D ;4)`→`(A ;8)`→`(E ;8)`

b On fusionne (fonction `fusion`) les deux feuilles les plus légères (les deux premières) pour en faire un arbre ; on crée un nouveau noeud qui a pour information la chaine `GH` et pour poids 2, son fils gauche est l'arbre `(G ;1)`, et son fils droit l'arbre `(H ;1)`. Notre liste est devenue :
   `lst`→`(F ;2)`→`(GH ;2)`→`(B ;4)`→`(C ;4)`→`(D ;4)`→`(A ;8)`→`(E ;8)`
   On recommence la fusion des deux premiers arbres, et la liste devient :
   `lst`→`(B ;4)`→`(C ;4)`→`(D ;4)`→`(FGH ;4)`→`(A ;8)`→`(E ;8)`
   Puis :
   `lst`→`(D ;4)`→`(FGH ;4)`→`(A ;8)`→`(E ;8)`→`(BC ;8)`
   Puis :
   `lst`→`(A ;8)`→`(E ;8)`→`(BC ;8)`→`(DFGH ;8)`
   Puis :
   `lst`→`(BC ;8)`→`(DFGH ;8)`→`(AE ;16)`
   Puis :
   `lst`→`(AE ;16)`→`(BCDFGH ;16)`
   Enfin :
   `lst`→`(AEBCDFGH ;32)`
   L'unique arbre que contient la liste est l'arbre de Huffman recherché. une fosi l'arbre construit, il suffit pour obtenir le code d'un caractère de descendre la racine jusqu'à la feuille le contenant, en notant le chemin suivi. Pour diriger la descente, chaque noeud contient les caractères situés en dessous de lui (champs `chaine`).

   Pour décoder un message, on part de la racine de l'arbre de Huffman, et l'on suit le chamin indiqué par les bits jusqu'à ce qu'on arrive sur une feuille. On a alors décodé le caractère porté par cette feuille. On poursuite le décodage du reste du message en repartant de la racine.

Notations et remarques :

La valeur constante `nb_maxchar` indique le nombre maximum de caractères que peut comporter le texte à compresser. La variable entière `nbcar` indique le nombre de caractères différents que comporte effectivement le texte à compresser. Dans tout le problème, le texte à compresser est représenté par la variable `txt` de type chaîne (`string`). Les variables chaîne `texte_code`

et `texte_decode` contiennent respectivement le texte compressé et le texte décompressé. Les tableaux d'entiers `frequence` et de caractères `car` contiennent respectivement les fréquences et les caractères qui sont présents dans le texte. Seulement les `nbcar` premières composantes de ces tableaux sont remplies.

Question 1 *En utilisant l'arbre ci-dessus, décoder le message 111110010111111101.*

Question 2 *Dans le cas où les `nbcar` caractères du texte à coder ont la même fréquence d'apparition, quelle est la profondeur de l'arbre de Huffman ?*

Question 3 *Écrire la procédure sans paramètre `initialisations` qui doit initialiser les tableaux `car` et `frequence` et la variable `nbcar`. Pour ce faire, la procédure doit examiner tous les caractères du texte à compresser `txt`.*

Question 4 *Écrire les deux versions, récursive et itérative, de la fonction booléenne `appartient` qui teste si un caractère `c` est présent ou non dans une chaîne `ch`.*

Question 5 *Écrire la fcontion `insere` qui renvoie la liste obtenueaprès avoir inséré un arbre dans une liste d'arbres ordonnée suivant l'ordre croissant des champs `poids` des noeud racines. La liste résultat doit respecter cette condition.*

Question 6 *En utilisant la fonction précédente, écrire la fonction `creer_liste` qui renvoie comme résultat la liste des `nbcar` arbres réduits à un seul noeud. La liste doit être classée par ordre croissant des champs `poids` des noeuds. Cette fonction utilise les tableaux `car` et `frequence` qui sont des variables globales.*

Question 7 *Écrire la fonction `fusion` qui prend pour argument deux variables `a1` et `a2` et renvoie comme résultat un nouvel arbre `a` tel que :*
- *le champ `poids` de `a` est la somem des champs `poids` des noeuds racines des arbres `a1` et `a2`.*
- *Le champ `chaine` de `a` est la concaténation des champs `chaine` des noeuds racines des arbres `a1` et `a2`.*
- *le champ `fg` de `a` est l'arbre `a1`.*
- *le champ `fd` de `a` est l'arbre `a2`.*

Question 8 *Écrire la fonction `creer_arbre` qui prend pour argument une liste ordonnée d'arbres et renvoie comme résultat l'arbre de Huffman. Il s'agit ici de programmer l'algorithme de construciton de l'arbre de Huffman.*

Question 9 *Écrire la fonction `codage` qui prend pour argument une chaîne `ch` et qui renvoie comme résultat la chaîne codée, c'est à dire que chaque caractère de la chaîne est remplacé par son code de Huffman (suite de 0 et de 1). La variable globale `huff` désigne l'arbre de Huffman.*

Question 10 *Écrire la fonction `decodage` qui prend en argument une chaine correspodnant à un texte codé et qui renvoie comme résultat la chaîne décodée.*

En fait, l'arbre de Huffman sert principalement pour la construciton du code et pour le décodage. Une fois construit comme on vient de le voir, on peut remplir un tableau de chaînes `huff_code` qui contient le code de Huffman du i-ème caractère du tableau `car`. Pour le décodage, l'arbre utilisé est «moins encombrant» que celui que l'on vient de construire : les champs `chaine` et `poids` ne sont plus créés.

Pour connaître le code d'un caractère, on ne parcourt plus l'arbre, il suffit de rechercher son rang dans le tableau `car`, pour ensuite récupérer son code dans el tableau `huff_code`. Pour accélérer cette recherche, on classe (par ordre croissant par exemple) le tableau `car` de manière à pouvoir effectuer une recherche dichotomique.

Question 11 *Écrire la fonction `recherche` qui renvoie comme résultat un nombre entier, qui indique le range où est placé le caractère `c` dans le tableau classé `t`. On demande, dans cette question, de programmer l'algorithme de recherche dichotomique.*

Question 12 *Quelle est la complexité de l'algorithme précédent ?*

Question 13 *Écrire la procédure `parcours` qui parcourt l'arbre de Huffman une et une seule fois, de manière à ce que lorsqu'une feuille est atteinte, la variable globale `code` contienne le code de l'unique caractère «étiquetant» cette feuille. La valeur de la variable `code` est alors rangée dans le tableau `huff_code` à l'endroit voulu.*

Annexe CAML

Types et fonction globales :

```
let txt = ref (read_line()) ; ;
type paire = (poids : int, chaine :string) ; ;
type arbre =
    nil
  | noeud of arbre * paire * arbre ; ;
let nbcar_max = 80 ; ;
let car = make_vect nbcar_max ' ' and
    frequence = make_vect nbcar_max 0 and
    huff_code = make_vect nbcar_max "" and
    nbcar = ref (0) and
    texte_code = ref ("") and
    texte_decode = ref ("") and
    huff = ref (nil) and
    code = ref ("") ; ;
```

Liste des fonctions demandées :

```
initialisations : unit → unit = <fun>
appartient : char → string → bool = <fun>
insere : arbre → arbre list → arbre list = <fun>
creer_liste : unit → arbre list ref = <fun>
fusion : arbre → arbre → arbre = <fun>
creer_arbre : arbre list ref = <fun>
codage : string → string = <fun>
decodage : string → string = <fun>
cherche : string → string → int = <fun>
parcours : arbre → unit = <fun>
```