Option MP

EPREUVE D'INFORMATIQUE

Durée : 4 heures

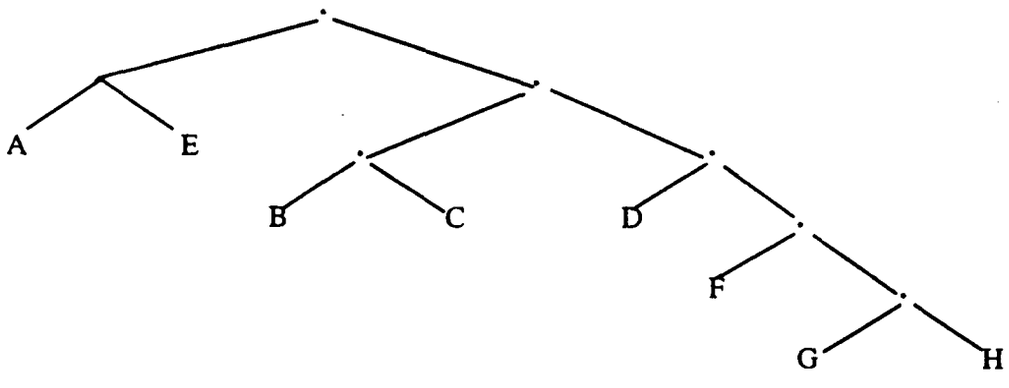
Les algorithmes demandés peuvent être codés en CAML ou en PASCAL. Quand aucune directive n'est donnée le candidat peut établir une version récursive ou non. Quel que soit le choix du langage, les algorithmes doivent être écrits de la manière la plus courte possible, parfaitement lisible, avec une indentation convenable, sans aucune rature et en ***respectant scrupuleusement les notations introduites****. Ils doivent être documentés par des explications concises et précises sur les points qui le nécessitent. La lecture de l'annexe correspondant au langage choisi est vivement conseillée avant d'aborder les questions de programmation.*
Les réponses qui ne respecteraient pas les consignes précédentes ne seront pas prises en considération. Les questions sont assez souvent indépendantes les unes des autres, il est cependant conseillé de respecter la chronologie proposée.

Préliminaire

Le problème est consacré à l'étude d'algorithmes de compression de données. Ces algorithmes sont particulièrement utilisés dans le domaine des transmissions ou des télécommunications où ils sont la source d'un gain de temps considérable. Ils restent le plus sûr moyen d'économiser la mémoire de masse.

I- L'arbre de HUFFMAN

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 des 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 feuilles 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 coder les huit premières lettres de l'alphabet.



Si l'on convient de représenter les branches gauches 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 le code de A est 00; celui de B est 100; celui de C est 101; etc...
Chaque noeud de l'arbre contient comme information 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 construction 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 constituée 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 les 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 chaîne 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 fois l'arbre construit, il suffit pour obtenir le code d'un caractère de descendre de 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 chemin indiqué par les bits jusqu'à ce qu'on arrive sur une feuille. On a alors décodé le caractère porté par cette feuille. On poursuit le décodage du reste du message en repartant de la racine.

Notations et remarques:
La valeur constante `nbcar_max` indique le nombre maximum de caractères différents 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 variable `txt` de type chaîne. Les variables chaînes `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.
Dans les questions suivantes, on demande parfois d'écrire des fonctions ou des procédures. Ces deux notions ont un sens précis en PASCAL, mais en CAML la notion de procédure n'existe pas. En CAML, on peut assimiler cette notion à celle d'une fonction qui se contente de n'exécuter qu'une série de commandes, qui produisent des effets (modifications de l'état courant) sans produire de résultat au sens mathématique du terme (fonction CAML dont le résultat est de type `unit`).

1) En utilisant l'arbre ci-dessus, décoder le message: 1111100101111101.

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

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

4) Ecrire 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`.

5) Ecrire la fonction `insere` qui renvoie la liste obtenue après avoir inséré un arbre dans une liste d'arbres ordonnée suivant l'ordre croissant des champs `poids` des noeuds racines. La liste résultat doit respecter cette condition.

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 suivant l'ordre croissant des champs `poids` des noeuds. Cette fonction utilise les tableaux `car` et `frequence` qui sont des variables globales.

7) Ecrire la fonction `fusion` qui prend pour argument deux arbres `a1` et `a2` et renvoie comme résultat un nouvel arbre *a* tel que :
Le champ `poids` de `a` est la somme 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`.

8) Ecrire 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 construction de l'arbre de Huffman.

9) Ecrire la fonction `codage` qui prend pour argument une chaîne `ch` et 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). **Attention**, les 0 et 1 qui sont dans la réalité codés sur un bit sont, dans ce problème, codés sur un octet. La variable globale `huff` désigne l'arbre de Huffman.

10) Ecrire la fonction `decodage` qui prend en argument une chaîne correspondant à un texte codé et qui renvoie comme résultat la chaîne décodée.

En fait, l'arbre de Huffman sert principalement pour la construction du code et pour le décodage. Une fois construit comme on vient de le voir, on peut remplir un tableau de chaînes (`huff_code`) en parcourant entièrement l'arbre une seule fois. La ième composante du tableau `huff_code` contient le code de Huffman du ième 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 le 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.

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

12) Quelle est la complexité de l'algorithme précédent?

13) Ecrire 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.

II- Code de SHANNON-FANO

C'est aussi un code à longueur variable qui est très facilement décodable. Voici la façon de le construire:
- On ordonne les caractères suivant l'ordre croissant de leur fréquence.
- On cherche ensuite à partitionner le tableau de caractères en deux sous-tableaux de manière à ce que les sommes des fréquences des caractères de chacun de ces sous-tableaux soient égales ou le « plus proches possible ». Tous les codes des caractères du premier sous-tableau commencent par un 0 et ceux du deuxième sous-tableau par un 1.
- Puis on continue à partitionner chacun des deux sous-tableaux en deux, en rajoutant un 0 aux codes des caractères des premiers sous-tableaux et en rajoutant un 1 aux codes des caractères des deuxièmes sous-tableaux. On continue ainsi les partitions jusqu'à ce qu'on aboutisse à un sous-tableau réduit à un seul caractère.

Plus précisemment un sous-tableau noté G_{d_f} est l'ensemble des composantes du tableau *car* dont les indices sont compris entre les deux nombres entiers d et f. Si S_{d_f} désigne la somme des fréquences affectées à ces composantes, on cherche la partition G_{d_k}, G_{k+1_f} du sous-tableau G_{d_f} telle que $\left|S_{d_k} - S_{k+1_f}\right|$ soit *minimum*.

14) On suppose que les caractères sont déjà classés suivant l'ordre croissant de leur fréquence d'apparition, écrire la procédure `shannon_fano` qui remplit le tableau de chaînes `sf_code` avec le code de Shannon-Fano des `nbcar` caractères.

15) Ecrire le code de Shannon-Fano et le code de Huffman des six premières lettres de l'alphabet dans le cas où le texte à compresser est égal à AAAABCDEFG.

16) Quel est le nombre moyen de bits nécessaires au codage d'un caractère dans chacun des deux cas de la question précédente? Peut-on en tirer une conclusion?

ANNEXE PASCAL

Constante, type et variables globales:

```
const
        nbcar_max = 80;

type
           str = string[nbcar_max];
       tab_car = array [1..nbcar_max] of char;
       tab_int = array [1..nbcar_max] of integer;
       tab_car = array [1..nbcar_max] of str;

         paire = record
                    poids : integer;
                   chaine : str
                 end;

         arbre = ^noeud;

         noeud = record
                     info : paire;
                   fg,fd : arbre
                 end;

      pdoublet = ^doublet;
       doublet = record
                       rac : arbre;
                      suiv : pdoublet
                 end;

var
                        frequence : tab_int;
                              car : tab_car;
                  sf_code,huff_code : tab_str;
                             nbcar : integer;
                              code : str;
     txt,texte_code,texte_decode : string;
                             huff : arbre;
                                i : integer;
```

Les types de données `pdoublet` et `doublet` servent à représenter une liste chaînée d'arbres:
Le champ `rac` désigne un arbre, le champ `suiv` permet d'accéder à l'élément suivant de la liste. Si `x` pointe vers un doublet de la liste, on écrit `x^.rac^.info.poids` pour désigner le `poids` associé au champ `info` de la racine de l'arbre désigné par le champ `rac` du doublet pointé par `x`.
La liste `lst` → (G;1) → (H;1) → (F;2) → ... → (E;8) introduite dans la présentation de l'algorithme est codée avec cette structure:
La valeur `x^.suiv` est symbolisé par → et la valeur `x^.rac` est représentée par (U;V).
Le couple (U;V) symbolise le noeud racine d'un arbre où U=`x^.rac^.info.poids` et
V= `x^.rac^.info.chaine`, les champs `fg` (fils gauche) et `fd` (fils droit) ne sont pas représentés sur le schéma.

En-tête des procédures ou fonctions demandées:

```
procedure initialisations;

function appartient(c:char;ch:str):boolean;

function insere(d,lst:pdoublet):pdoublet;

function creer_liste:pdoublet;

function fusion(a1,a2:arbre):arbre;

function creer_arbre(lst:pdoublet):arbre;

function codage(ch:string):string;

function decodage(ch:string):string;

function cherche(.....):integer;

procedure parcours(a:arbre);

procedure shannon(.....);
```

Quelques fonctions de la bibliothèque PASCAL:

`ord(x:char)` : renvoie le code ASCII du caractère `x`.

`chr(x:byte)` : renvoie le caractère dont la valeur ordinale (code ASCCI) est `x`.

`length(s:string)` : renvoie la longueur de la chaîne `s`.

`copy(s:string;pos:integer;long:integer)` : renvoie la sous-chaîne extraite de `s` à partir de la position `pos` et qui possède `long` caractères.

`concat(s1,...,sn:string)` : revoie une chaîne qui est la concaténée des chaînes `s1`,..., `sn`.
On peut également utiliser l'opérateur binaire de concaténation, noté +, pour concaténer des chaînes de caractères.

ANNEXE CAML

Types et valeurs 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
         sf_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 → arbre = <fun>
codage          : string → string = <fun>
decodage        : string → string =<fun>
cherche         : ..... →int = <fun>
parcours        : arbre → unit = <fun>
shannon         : ..... →unit = <fun>
```

Quelques fonctions de la bibliothèque CAML:

`int_of_char` : `char → int` renvoie le code ASCII de son argument.
`char_of_int` : `int → char` renvoie le caractère de code ASCII donné.
`char_for_read` : `char → string` renvoie une chaîne représentant le caractère donné.
`string_length` : `string → int` renvoie la longueur de la chaîne donnée.
`nth_char` : `string → int → char`, `nth_char s n` renvoie le caractère numéro `n` de la chaîne `s`.
`sub_string` : `string → int → int → string`, `sub_string s d l` renvoie une nouvelle chaîne de longueur `l` contenant les caractères de la chaîne `s` à partir du caractère numéro `d`.
`préfix ^` : `string → string → string`, `s1^s2` renvoie une nouvelle chaîne contenant la concaténation des chaînes `s1` et `s2`.
`list_length` : `'a list →int` renvoie le nombre d'éléments de la liste donnée.
`hd` : `'a list → 'a` renvoie le premier élément de la liste donnée.
`tl` : `'a list → 'a list` renvoie la liste donnée privée de son premier élément.

www.tunisie-etudes.info

Ce document a été téléchargé depuis
www.tunisie-etudes.info

Des documents gratuits, devoirs, examens, cours, exercices, corrigés... Ainsi que toute une rubrique pour vous aider à trouver un emploi sans oublier les avis de concours en direct

Notre page Twitter :
http://www.twitter.com/TunisieEtudes
Notre page FaceBook :
http://www.facebook.com/TunisieEtudes



Merci d'avoir choisi www.tunisie-etudes.info
Bonne lecture et bon travail