Université Lille1 Sciences et Technologies

Info 204 - ASD

2010–2011
Licence STS - Semestre 4

DS2 - documents de cours, TD, TP autorisés

A lire impérativement avant de commencer

Lorsque, dans une question, est demandée la complexité, c'est une fonction de la taille de la donnée qui est attendue. Lorsque c'est le comportement asymptotique, la réponse attendue est soit une notation $\mathcal{O}$, soit une notation Θ soit une notation Ω.

Vous pourrez utiliser toute fonction vue en cours, TD, TP à la condition expresse de spécifier correctement les entrées et les sorties.

Le code des fonctions peut être écrit soit en OCaml, soit en pseudocode mais de manière très claire.

Les temps indiqués pour chaque exercice sont approximatifs et seulement là pour vous aider à répartir votre temps de travail.

Exercice 1 : AVL - 15 minutes

On considère ici des AVL d'entiers déclarés ainsi :

```
type avl = Vide | Cons of noeud
and noeud = { valeur : int; mutable droit : avl; mutable gauche : avl }
```

Q 1.1 Dessiner l'AVL obtenu après chaque insertion des valeurs suivantes, dans cet ordre : 6, 4, 5, 2, 1, 3, 7, 0.

Q 1.2 Dessiner toutes les étapes de la suppression de la valeur 3.

Exercice 2 : Un tri étrange - 20 minutes

Nous présentons ci-dessous un algorithme de tri :

```
let rec tri t i j =
  if j > i then
    if i + 1 = j then begin
      if t.(i) > t.(j) then
        (* echange dans t les valeurs aux indices i et j *)
        echanger t i j;
    end else begin
      tri t i (j - 1);
      tri t (i + 1) j;
      tri t i (j - 1);
    end
```

Q 2.1 Déroulez l'appel à `tri [| 4; 1; 3; 2 |] 0 3`. Vous indiquerez les valeurs de `i` et `j` à chaque appel et les valeurs de `t`.

Q 2.2 Pourquoi, parmi les trois appels recursifs à `tri`, le dernier appel à la procédure de tri est-il nécessaire ? Donnez une permutation de taille 3 où, sans cet appel, l'algorithme ne fonctionne pas.

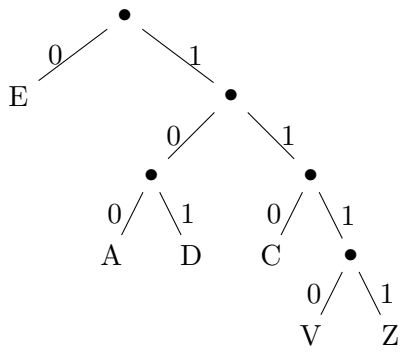
Q 2.3 Etablissez l'équation de récurrence de la complexité en temps de l'algorithme.

Q 2.4 Donnez le comportement asymptotique en temps. Justifiez.

Exercice 3 : Arbre de codage - 30 minutes

Chaque feuille d'un arbre binaire correspond à un unique chemin de la racine à celle-ci. On peut ainsi associer un code à chaque feuille de l'arbre. La procédure est simple, en partant de la racine on crée une chaîne de caractères à laquelle on concatène un 0 si on emprunte le fils gauche et un 1 si on emprunte le fils droit.

Nous nous intéressons à des arbres dont les feuilles contiennent une lettre et aucune feuille ne contient la même lettre, ainsi il n'y a pas d'ambigüité. En suivant le chemin menant de la racine à une feuille et la procédure de codage décrite, on obtient le code associé à cette feuille. Sur l'exemple suivant, la lettre `C` aura pour code `110` :



Q 3.1 Donnez le code associé à chaque feuille de l'arbre ci-dessus.

Il est facile de voir qu'il existe une bijection entre un code et un caractère. Cela signifie qu'à partir d'une suite de 0 et de 1 on obtient une et une seule lettre (et réciproquement). Une chaîne de caractères peut donc être codée de manière unique en associant à chaque lettre son code et en concaténant les codes.

Q 3.2 Donnez le mot associé au codage `101100101100`.

Q 3.3 Donnez le code associé au mot `EVADE`.

On utilisera le type suivant :

```
type arbre = Vide | Cons of noeud
and noeud = { valeur : char; mutable droit : arbre; mutable gauche : arbre }
```

et la valeur n'aura de signification que pour les feuilles.

Q 3.4 Ecrire une fonction `decode : string -> arbre -> string` qui transforme une chaîne formée de `0` et de `1` en le mot associé suivant les codes donnés par l'arbre passé en paramètre.

Q 3.5 Quelle est le comportement asymptotique en temps de cette fonction en fonction de la longueur de la chaîne passée en paramètre à `decode` ?

Q 3.6 Décrivez une méthode en français, la plus efficace possible en temps, pour la fonction inverse, `code`, qui réalise le codage d'une chaîne de caractères `s` en une suite de `0` et de `1` en utilisant un arbre `a` *qui ne contient que les lettres du mot à coder.*

Q 3.7 Quelle est son comportement asymptotique en temps ?

Exercice 4 : Itérateurs d'arbres - 45 minutes

On définit le type d'arbre binaire suivant :

```
type binaire = Vide | Cons of noeud
and noeud = { valeur : int; droit : binaire; gauche : binaire}
and iterateur = { mutable pere : binaire Stack.t; mutable courant : binaire }
```

ainsi que trois exceptions : `ArbreVide, IterateurEnFeuille, IterateurEnRacine`.

On souhaite mettre en place des itérateurs pour permettre le parcours de ces arbres. *On supposera que les arbres manipulés contiennent au moins une valeur.*

Q 4.1 Expliquer brièvement pourquoi le champ `pere` est une pile.

Q 4.2 Donner le code de la fonction fournissant un nouvel itérateur positionné sur la racine de l'arbre passé en paramètre.

Q 4.3 Donner le code de la fonction permettant de descendre sur le fils gauche du nœud courant de l'itérateur qui lèvera l'exception `IterateurEnFeuille` si le nœud courant de l'itérateur est une feuille.

Q 4.4 Donner le code de la fonction permettant de remonter sur le père du nœud courant de l'itérateur qui lèvera l'exception `IterateurEnRacine` si le nœud courant de l'itérateur est la racine.

On supposera disposer également d'une fonction permettant de descendre à droite, et d'une fonction `valeur` permettant d'obtenir la valeur associée au nœud courant de l'itérateur qui lève l'exception `ArbreVide` si le courant est vide.

Q 4.5 Ecrire une fonction utilisant **uniquement** les primitives sur les itérateurs et la gestion des exceptions pour réaliser une impression préfixée de l'arbre. Un exemple d'affichage est donné ci-dessous :