Examen Module IF122

Université Paris 7 - Denis Diderot

06 Juin 2006

Durée : 3 heures.
Documents : Documents manuscrits (notes de cours et de TD/TP) autorisés.
NB : (1) Toutes les réponses doivent être clairement justifiées. (2) Il est *fortement conseillé de lire le sujet en entier* avant de commencer à répondre aux questions.

Exercice 1

Question 1 : Donner l'arbre binaire de recherche construit en insérant (en utilisant l'algorithme d'insertion vu en cours) successivement les éléments de la séquence d'entiers suivante :

$$100, 51, 25, 200, 30, 86, 210, 150, 7$$

Noter bien que les insertions se font selon l'ordre défini par la séquence, c'est-à-dire, en commençant par l'élément le plus à gauche (ici 100) et terminant par l'élément le plus à droite (ici 7).

Question 2 : Est-ce que l'on obtient le même arbre si l'on inverse l'ordre d'insertion des éléments 51 et 25 ? Justifier la réponse.

Question 3 : Est-ce qu'il existe une séquence différente de celle donnée ci-dessus, mais qui permet de construire exactement le même arbre binaire de recherche ? Justifier la réponse.

Question 4 : Donner un autre ordre des éléments de la séquence ci-dessus de sorte que l'arbre de recherche construit selon ce nouvel ordre a la propriété suivante : chaque sommet de l'arbre a un fils gauche vide. Même question pour construire un arbre où chaque sommet a un fils droit vide.

Exercice 2

On appelle *facteur* d'une suite finie $x_1 \dots x_n$, toute sous-suite $x_i x_{i+1} \dots x_j$ d'éléments consécutifs. On suppose ici que les éléments sont des entiers, et que les suites d'entiers sont représentées par des listes chaînées. On suppose donnée la classe des liste chaînées (d'entiers).

Question : Ajouter à la classe des listes chaînées une fonction

```
public static ListeEnt Simplifier(ListeEnt l)
```

qui prend comme argument une liste chainée l et retourne une (nouvelle) liste correspondant à une séquence d'éléments obtenue de la manière suivante : chaque facteur de la séquence représentée par l composé d'un seul et même élément est remplacé par une seule occurrence de cet élément (c'est-à-dire que la fonction retourne une liste où les répétitions successives d'un même élément sont supprimées). Par exemple si la liste initiale l contient la séquence $2,5,5,5,2,2,1,1,1,1,5,1,1$ alors le résultat est une liste qui correspond à la suite $2,5,2,1,5,1$.

Exercice 3

Question 1 : Ecrire une fonction **récursive** *InverserListe* qui, étant donnée une liste chaînée (donnée en paramètre), construit et retourne une (nouvelle) liste qui contient les éléments de la première liste dans l'ordre inverse. Par exemple si la séquence des éléments représentés dans la liste donnée est c,a,b,a,c,d, alors la liste construite par *InverserListe* correspond à la séquence d,c,a,b,a,c.

Etant donné un entier naturel n, la représentation de n en base 2 est définie comme étant la séquence $a_k,a_{k-1},\ldots,a_0$ de 0 ou de 1, telle que :

$$n = a_k \cdot 2^k + a_{k-1} \cdot 2^{k-1} + \cdots + a_1 \cdot 2 + a_0 \cdot 2^0 \quad (1)$$

Par exemple, l'entier 3 a pour représentation en base 2 la séquence $1,1$ puisque $3 = 1 \cdot 2 + 1 \cdot 1$ (on rappelle que $2^0 = 1$). De même, les codes en base 2 des entiers 4 et 6 sont respectivement $1,0,0$ et $1,1,0$.

Question 2 : Ecrire une fonction **récursive** *Décoder* qui étant donnée $\sigma = a_k,a_{k-1},\ldots,a_0$ une suite de 0 et de 1 représentée par une liste chaînée, calcule l'entier naturel n dont le code en base 2 correspond à σ. On considère que si la séquence donnée σ est vide, alors la fonction *Décoder* doit rendre la valeur 0.

Indication : Utiliser le fait que l'égalité (1) ci-dessus est équivalente à :

$$n = a_0 + 2 \cdot (a_1 + 2 \cdot (a_2 + 2 \cdot (\cdots a_{k-2} + 2 \cdot (a_{k-1} + 2 \cdot a_k)) \cdots)))$$

Exercice 4

On considère la classe *ArbreBin* des arbres binaires dont les sommets stockent des chaînes de caractères (des string). On suppose données les fonctions suivantes (il ne faut pas les implémenter) :

1. `string Valeur()` - retourne la valeur stockée dans le sommet (dans la racine),
2. `boolean EstVide()` – retourne `true` si et seulement si l'arbre est vide,
3. `Arbre FilsGauche()` et `Arbre FilsDroit()` retournent respectivement les sous arbres gauche et droit, si l'arbre courant n'est pas vide (sinon ces fonctions ne sont pas définies).

On suppose que les arbres binaires manipulés dans cet exercice représentent des expressions arithmétiques. Les feuilles de ces arbres sont étiquetées par des chaînes représentant des entiers naturels, et leurs sommets internes sont étiquetés par des chaînes représentant des opérateurs binaires sur les entiers. On considère qu'un opérateur peut être soit celui de l'addition, soit celui de la soustraction, soit celui de la multiplication.

On admet que la classe ArbreBin est déjà équipée des fonctions statiques `EstEntier`, `EstAdd`, `EstSoust`, `EstMult`, `Conversion` qui prennent toutes en paramètre une string *s* et dont la valeur retournée est définie comme suit :

- `EstEntier` : retourne true si et seulement si *s* représente un nombre entier
- `EstAdd` : retourne true si et seulement si *s* est égale à "+"
- `EstSoust` : retourne true si et seulement si *s* est égale à "-"
- `EstMult` : retourne true si et seulement si *s* est égale à "*"
- `Conversion` : si *s* représente un entier, alors cette fonction retourne sa valeur. Par exemple, si *s* est la string "12" alors `Conversion` retournera l'entier 12.

Question 1 : Ajouter dans la classe `ArbreBin` la fonction

```
public static int Valeur(ArbreBin a)
```

qui calcule et retourne la valeur de l'expression arithmétique représentée par l'arbre `a`.

Il est bien connu que les opérations d'addition et de multiplication sont commutatives, mais que la soustraction ne l'est pas. Deux expressions arithmétiques sont dites *équivalentes modulo commutativité* si et seulement si l'une peut être obtenue à partir de l'autre en utilisant (uniquement) les règles de la commutativité.

Par exemple, l'expression $(((3+6)+4)*(5-(2*(7+1))))$ est équivalente modulo commutativité à l'expression $(5-(2*(1+7))*((6+3)+4))$. Ces expressions ne sont pas équivalentes modulo commutativité à $(5-(2*(1+7))*(3+(6+4)))$. (En effet, pour obtenir l'équivalence avec cette dernière expression, il faut se servir de la règle de l'associativité pour le $+$.)

Question 2 : Ecrire une fonction

```
public static bool EquivC (ArbreBin a, ArbreBin b)
```

qui prend en paramètres deux arbres binaires représentant deux expressions arithmétiques, et retourne true si ces deux expressions sont équivalentes modulo commutativité.

Exercice 5

On considère dans cet exercice des arbres binaires de recherche qui contiennent des entiers (double). On suppose que ces arbres sont implémentées par une classe `ABR` qui fournit à l'utilisateur les méthodes suivantes (ne pas les implémenter) :

1. `double Valeur()` - retourne la valeur stockée dans le sommet (dans la racine),
2. `boolean EstVide()` – retourne `true` si et seulement si l'arbre est vide,
3. `ABR FilsGauche()` et `ABR FilsDroit()` retournent respectivement les sous arbres gauche et droit, si l'arbre courant n'est pas vide (sinon ces fonctions ne sont pas définies).

Question 1 : Ecrire une fonction récursive `somme` qui prend comme arguments deux nombres de type `double` : `x` et `y` et retourne la somme de tous les valeurs *v* stockées dans l'arbre binaire de recherche telles que $\texttt{x} \leq v \leq \texttt{y}$. (Si l'arbre ne contient aucune valeur dans l'intervalle [x,y] alors la fonction retourne 0.)

NB : Si la fonction `somme` est statique alors l'arbre `a` passe aussi comme un argument, sinon l'arbre est accessible comme l'objet `this` dans la fonction.

Attention : Votre fonction doit éviter les appels récursifs inutiles, par exemple si la valeur stockée à la racine de `a` est inférieure ou égale à `x` alors qu'est-ce que nous pouvons dire des valeurs stockées dans le sous arbre gauche ?

Question 2 : Ecrire une fonction récursive qui affiche les éléments de l'arbre (binaire de recherche) dont la valeur est dans l'interval $[x, y]$, dans l'ordre *décroissant* de ces valeurs.