Université de Nice-Sophia Antipolis — vendredi 31 mai 2013
POLYTECH PeiP2

DS d'Algorithmique et Programmation Java

Durée: 1h30
Documents non autorisés

Note : Les 4 parties de ce DS sont indépendantes. La qualité des commentaires, avec notamment la présence d'affirmations significatives, ainsi que les noms donnés aux variables, l'emploi à bon escient des majuscules et la bonne indentation rentreront pour une part importante dans l'appréciation du travail.

---

1 Pile

---

1) La classe `PileTableau`, vue en TD, a comme principal inconvénient le fait de limiter la taille de la pile et d'avoir à gérer la notion de *pile pleine*. Récrivez la méthode `empiler` de façon à agrandir le tableau `lesÉléments` lorsqu'il est plein pour augmenter la taille de la pile.

On rappelle l'en-tête de cette méthode :

```
/**
 *  Rôle : empile dans la Pile courante l'objet x
 *
 *  @param x l'objet de type T générique à empiler
 */
public void empiler(T x)
```

---

2 Le minimum excludant d'un ensemble (mex)

On rappelle que le *minimum excludant* d'un ensemble d'entiers positifs est le plus petit entier qui n'est pas dans l'ensemble :

$$mex(E) = \min_{n \geq 0} \{n | n \notin E\}$$

Par exemple :

$$mex(0,1,2,4) = 3$$
$$mex(1,3,4,5) = 0$$
$$mex(0,1,2,3,4) = 5$$
$$mex() = 0$$

2) L'ensemble d'entiers positifs est représenté par une liste non ordonnée d'entiers. Écrivez une méthode qui prend en paramètre donné une telle liste d'entiers et qui retourne son *mex*.

3) Quelle est la complexité de cette méthode ?

4) Si on suppose que la liste est ordonnée (ordre croissant), est-il possible d'écrire une méthode plus efficace ? Donnez alors sa complexité.

Rappel : ci-dessous l'interface `Liste`

```
public interface Liste<T> {
  public int longueur();
  public T ième(int r) throws RangInvalideException;
  public T insérer(int r, T e) throws RangInvalideException;
  public T supprimer(int r) throws RangInvalideException;
  public T échanger(int r1, r2) throws RangInvalideException;
}
```

en outre, le rang de la première valeur d'une liste est 1.

---

3 Tri

Le but de cet exercice est de trier une liste d'entiers.

5) Écrivez une méthode de classe `estTrié` qui retourne `true` si et seulement si l'objet de type `Liste<Integer>` passé en paramètre est trié de façon croissante.

6) Écrivez maintenant la méthode `trier` qui trie une liste de n entiers comme suit : on tire 2 rangs différents au hasard (compris entre 1 et n), on échange les deux éléments associés, et on regarde si le tableau résultant est trié. Si le tableau est trié, la méthode s'arrête, sinon, elle recommence.

7) Est-ce que ce tri peut se faire dans un temps borné pour toute valeur n ? D'après vous, quelle est la taille de la liste à ne pas dépasser pour avoir un temps d'exécution raisonnable ?

---

4 Arbre binaire

---

8) Dans la classe `ArbreBinaireChaîné`, ajoutez la méthode `boolean estFeuille()` définie dans l'interface `ArbreBinaire`. Si la méthode est appelée sur un objet instance de `ArbreBinaireChaîné`, elle renvoie **true** si cet arbre (l'arbre courant) est une feuille, et **false** sinon. Remarquez que pour simplifier cette question et la suite de l'exercice, la méthode `estFeuille` ne renvoie pas l'exception `ArbreVideException` et qu'on supposera donc que cette méthode n'est jamais appelée sur un arbre qui serait vide.

9) On ajoute à l'interface `ArbreBinaire` la méthode `int nbFeuilles()` qui renvoie le nombre de feuilles de l'arbre courant. Cet arbre n'étant donc pas réduit forcément à une simple feuille, il va falloir utiliser un parcours de tout l'arbre afin d'aller compter les feuilles de son sous-arbre gauche et de son sous-arbre droit, s'ils ne sont pas des arbres vides évidemment. Écrivez le code de cette méthode dans la classe `ArbreBinaireChaîné`.

10) De manière assez proche de la question précédente, on souhaite implémenter une méthode **int** `numéroterFeuilles(int num)`. Elle permet de numéroter les feuilles de l'arbre courant de gauche à droite (en supposant que la racine de l'arbre est en haut), à partir d'un numéro passé en paramètre (le paramètre nommé `num`) Cette méthode renvoie aussi un entier qui correspond au numéro à utiliser si l'on veut continuer à numéroter un autre arbre.

Par exemple, soit l'arbre donné ci-dessous, dont la structure est représentée par l'utilisation imbriquée de paranthèses :

(a (b) (c (d e)))

L'application de la méthode `numéroterFeuilles` donnera :

(a (b-1) (c (d-2 e-3))).

Remarquez qu'après avoir numéroté le sous-arbre gauche b, la méthode `numéroterFeuilles` doit renvoyer la valeur 2, car c'est à partir de cette valeur 2 que le sous-arbre droit de l'arbre dont la racine est a sera numéroté. On supposera que nous avons dans l'interface `ArbreBinaire` et la classe `ArbreBinaireChaîné` une méthode **void** `setNuméro(int n)` qui permet d'associer à la valeur de l'arbre courant le numéro n passé en paramètre.

Expliquez pourquoi cette méthode `numéroterFeuilles` doit absolument effectuer un parcours de l'arbre en profondeur et qui explore obligatoirement le sous-arbre gauche avant de s'intéresser au sous-arbre droit.