Université de Nice-Sophia Antipolis — POLYTECH
PeiP2 — 2017–2018

Examen de Structure de données

Durée : 0h30
Aucun document autorisé
Mobiles interdits

Nom : Prénom : Groupe 3

```
donnée/résultat  l : liste

pourtout i de 1 à longueur(l)-1 faire
   {Invariant : ..............              }
   pourtout j de longueur(l) à i+1 faire
      si clé(ième(l,j))<clé(ième(l,j-1)) alors
           aux ← ième(l,j)
           ième(l,j) ← ième(l,j-1)
           ième(l,j-1) ← aux
      finsi
   finpour
finpour
```

▶ 1. Expliquez ce que fait l'algorithme donné ci-dessus et comment il procède. Vous donnerez l'invariant et ses complexités (dans le meilleur, le pire, et le cas moyen).

Cet algorithme est l'algoirthme de tri à bulles (bubble sort). Il procède par échanges en faisiant remonter les éléments de clés les plus petites vers le début de la liste. L'invariant de boucle est :

```
{
 Invariant : la sous-liste de ième(l,1) à ième(l,i-1) est triée
             et les éléments de ième(l,i) à ième(l,n) sont
                supérieurs ou égaux à ceux de ième(l,1) à ième(l,i-1)
}
```

Pour une liste de longueur n, la complexité en nombre de comparaisons est la même pour le cas moyen, le meilleur et le pire : $\frac{1}{2}(n^2 - n) = \mathcal{O}(n^2)$.

la complexité en nombre d'échanges : dans le pire des cas, c'est-à-dire quand la liste est triée en ordre inverse, les éléments sont systématiquement échangés. Il y a $n-1$ échanges à la première étape, $n-2$ la seconde, etc. Puisqu'il y a $n-1$ étapes, le nombre d'échanges le pire est donc $\frac{1}{2}(n^2 - n) = \mathcal{O}(n^2)$. En revanche, il n'y a aucun échange si la liste est déjà triée.

.....

Une file d'attente est une suite d'éléments accessible **uniquement** par ses deux extrémités. On ajoute un élément par un coté (la queue) et on retire un élément par l'autre coté (la tête). C'est le comportement FIFO (First-In – First-Out).

On définit l'ensemble $\mathcal{F}ile$ des files d'attente formées d'éléments pris dans l'ensemble $\mathcal{E}$, sur lequel les opérations suivantes sont définies :

est-vide?	:	$\mathcal{F}ile$	$\rightarrow$	booléen
premier	:	$\mathcal{F}ile$	$\rightarrow$	$\mathcal{E}$
défiler	:	$\mathcal{F}ile$	$\rightarrow$	$\mathcal{F}ile$
enfiler	:	$\mathcal{F}ile \times \mathcal{E}$	$\rightarrow$	$\mathcal{F}ile$

Les axiomes qui décrivent la sémantique d'une file sont donnés ci-dessous. On considère que filevide $\in \mathcal{F}ile$.

$\forall f \in \mathcal{F}ile, \forall e \in \mathcal{E}$

1. est-vide?(filevide) = vrai
2. est-vide?(enfiler(f, e)) = faux
3. est-vide?$(f) \Rightarrow$ premier(enfiler(f, e)) $= e$
4. non est-vide?$(f) \Rightarrow$ premier(enfiler(f, e)) = premier(f)
5. est-vide?$(f) \Rightarrow$ défiler(enfiler(f, e)) = filevide
6. non est-vide?$(f) \Rightarrow$ défiler(enfiler(f, e)) = enfiler(défiler$(f), e)$
7. $\nexists f,\ f =$ défiler(filevide)
8. $\nexists e,\ e =$ premier(filevide)

▶ 2. À l'aide de la classe générique `LinkedList<T>` de l'API, écrivez en Java la classe générique `File<T>` qui implémente les 4 opérations précédentes.

```
import java.util.*;

public class File<T> {

    private List<T> l;

    public File() {
        l = new LinkedList<T>();
    }
    /**
     *  Rôle : renvoie vrai si la file courante est vide,
     *         et faux sinon
     */
    public boolean estVide() {
        return l.isEmpty();
    }
    /**
     *  Rôle : renvoie l'élément de tête de la file courante
     */
    public T premier() {
        assert !estVide();
        return l.get(0);
    }
    /**
     * Rôle : ajoute l'élément e en queue de la file courante
     */
    public void enfiler(T e) {
        l.add(e);
    }
    /**
     *  Rôle : supprime l'élément de tête de la file courante
     */
    public void défiler() {
        assert !estVide();
        l.remove(0);
    }
}
```

.....