TP n°6

Listes chaînées

1 Rappels

Une liste est une structure permettant de stocker des éléments, un peu la manière d'un tableau. Une liste est composée d'éléments, un élément étant constitué d'une valeur (valeur que l'on souhaite stocker) et d'une référence vers un autre élément. Une liste est donc une chaîne d'éléments, par exemple : $23 \to 3 \to 45 \to 3 \to 1 \to 3 \to 67$. Les classes `Liste` et `Element` sont constituées de la manière suivante.

```
public class Element{
  private int contenu;
  private Element suivant;
  ...
}

public class Liste{
  private int nombreElements;
  private Element tete;
  private Element fin;
  ...
}
```

2 Quelques opérations sur les listes

Exercice 1 Implémenter les classes `Liste` et `Element`. Ajouter dans chaque classe un constructeur par défaut et des constructeurs qui prennent des paramètres.

Exercice 2 Écrire les opérations élémentaires que vous avez vu en TD pour la classe `Liste` :
- `int tete()` qui renvoie le contenu de l'élément en tête ;
- `Liste queue()` qui renvoie la liste sans la tête ;
- `Boolean estVide()` qui teste si la liste est vide ;
- `int longueur()` qui retourne le nombre d'éléments ;
- `Liste ajouter(int x)` qui ajoute un élément au début, attention : si la liste était vide, il faut aussi modifier la valeur de `fin`.

Attention : Les méthodes `queue` et `ajouter` ne doivent pas modifier la liste passée en argument, mais en créer une nouvelle, de manière à ce que la nouvelle liste partage un bout avec l'ancienne.

Exercice 3 Écrire la méthode `String toString()` qui renvoie une chaîne de caractère contenant le contenu de la liste.

Exercice 4 Écrire une méthode `Liste ajouter(int k,int x)` qui ajoute l'élément `x` à la `k`-ième position de la liste, comme pour les tableaux on commence à compter à partir de la position 0.

Exercice 5 Écrire une méthode `Liste supprimer(int k)` qui enlève le `k`-ième élément de la liste. Faites attention en supprimant le premier ou le dernier élément, que se passe-t-il si la liste n'a qu'un seul élément ?

Exercice 6 Écrire dans la classe `Liste` une méthode itérative `int sommeI()` qui retourne la somme de tous les élément de la liste. Écrire une méthode récursive `int sommeR()` qui retourne la somme de tous les élément de la liste. Si n est le nombre d'éléments de la liste, quelle est la complexité de `sommeI` et de `sommeR` ?

Exercice 7 Écrire une méthode `int compter(int x)` qui renvoie le nombre d'occurrence de `x` dans la liste.

Exercice 8 Écrire une méthode `int position(int i)` qui, étant donné un entier, renvoie la position de sa première occurrence dans la liste ou -1 si il n'y est pas.

Exercice 9 Écrire une méthode `int minimum()` qui renvoie la valeur minimum de la liste.

Exercice 10 Soit `l'` est la liste obtenue de `l` en supprimant l'élément le plus petit, alors la liste triée de `l` est celle triée de `l'` à laquelle on a ajouté l'élément le plus petit de `l` au début. Écrire une méthode récursive `Liste trier()` qui trie une liste en s'appuyant sur cette remarque. Quelle est la complexité de cette méthode ?

3 Listes doublement chaînées

On souhaite maintenant créer des listes doublement chaînées, elles permettent de se déplacer dans deux directions, pour cela les éléments de la liste doivent pointer à la fois vers l'élément suivant et l'élément précèdent. Contrairement aux listes simplement chaînées, deux listes doublement chaînées différentes ne peuvent pas partager d'éléments, à partir de maintenant on modifiera donc les listes en place, ce qui évitera d'avoir à créer une nouvelle copie de la liste à chaque fois que l'on ajoute ou enlève un élément.

Exercice 11 Modifiez la classe `Element` afin d'obtenir des listes doublement chaînées. Ajoutez également des modifieurs pour ses propriétés.

Exercice 12 Ajoutez un constructeur `Liste(Liste l)` qui crée une copie de `l` en faisant un parcours en profondeur de `l`.

Exercice 13 Écrire une méthode `void supprimerDebut()` qui enlève l'élément début de la liste et une méthode `void supprimerFin()` qui enlève le dernier élément de la liste.

Exercice 14 Écrire une méthode `void ajouterDebut(int a)` qui ajoute un élément au début de la liste et une méthode `void ajouterFin(int a)` qui ajoute un élément à la fin de la liste.

Exercice 15 Écrire une méthode `void inverser()` qui inverse la liste : le premier élément devient le dernier et réciproquement.

Exercice 16 Écrire une méthode `void concatener(Liste l)` qui concatène une copie de la liste `l` à la fin de la liste courante.

Exercice 17 Écrire une méthode `void flipFlap()` qui inverse deux éléments consécutifs : le premier et le second, le troisième et le quatrième ... Par exemple $23 \rightleftarrows 3 \rightleftarrows 45 \rightleftarrows 2 \rightleftarrows 1 \rightleftarrows 4$ devient $3 \rightleftarrows 23 \rightleftarrows 2 \rightleftarrows 45 \rightleftarrows 4 \rightleftarrows 1$.

Exercice 18 Écrire une méthode `Liste fibonacci(int n)` qui renvoie la liste des n premiers éléments de la suite de Fibonacci. *Astuce* : se servir des deux éléments précédents dans la liste.

Pour rappel $F_0 = 0$, $F_1 = 1$ et $F_n = F_{n-2} + F_{n-1}$ si $n > 1$.