Algorithmique avancée - TD4

—o000o——o000o—

Listes circulaires et doublement chaînées

1 Liste croissante

On considère dans cet exercice une liste contenant des éléments de type `Double`.

1. Écrire la méthode `croissante()` qui retourne `true` si et seulement si la liste est triée par ordre croissant au sens large. On implémentera la fonction itérativement.
2. Proposer maintenant une version récursive de la méthode précédente nommée `croissanteRec()`. Vous pourrez considérer les cas de base suivants :
   - une liste vide est considérée comme triée par ordre croissant,
   - une liste avec un seul élément est croissante également ;

Solution :

```
1  package semaine4;
2
3  import linkedList.LList;
4  import linkedList.Node;
5
6  public class LListCrois extends LList<Double>{
7
8    public boolean croissante() {
9      Node<Double> p = this.getHead();
10     if (p == null || !p.hasNext()) return true;
11     else {
12       boolean croissante = p.getData() <= p.getNext().getData
           ();
13       while (croissante && p.getNext().hasNext()) {
14         p = p.getNext();
15         croissante = p.getData() <= p.getNext().getData();
16       }
17       return croissante;
18     }
19   }
20
21   public boolean croissanteRec() {
22     return croissanteNode(this.getHead());
```

```
23    }
24
25    private boolean croissanteNode(Node<Double> L) {
26      if (L == null || !L.hasNext()) return true;
27      else {
28        double premier = L.getData();
29        double second = L.getNext().getData();
30        return (premier <= second) && croissanteNode(L.getNext
              ());
31      }
32    }
33
34    public static void main(String[] args) {
35      LListCrois L = new LListCrois();
36      L.add(1.2); L.add(2.3); L.add(3.4); L.add(4.5);
37      System.out.println(L);
38      System.out.println(L.croissante());
39      System.out.println(L.croissanteRec());
40
41      LListCrois M = new LListCrois();
42      M.add(1.2); M.add(2.3); M.add(4.5); M.add(3.4);
43      System.out.println(M);
44      System.out.println(M.croissante());
45      System.out.println(M.croissanteRec());
46    }
47  }
```

java/LListCrois.java

2 Liste circulaire

On s'intéresse dans cet exercice à proposer des méthodes pour enrichir la barrière d'abstraction étudiée en cours.

1. Écrire un constructeur `LListCirc(LList L)` qui permet de construire une liste circulaire à partir d'une liste non circulaire.
2. Proposer une méthode `fusion(LListCirc<T> L)` qui permet de fusionner la liste circulaire `L` à la fin de la liste courante.
3. Écrire une méthode `remove(int index)` qui supprimer l'élément à l'indice passé en argument dans la liste circulaire. Tous les cas doivent pouvoir être traités.

Solution :

```java
package semaine4;

import linkedList.LList;
import linkedList.LListCirc;
import linkedList.Node;

public class LListCirc2<T> extends LListCirc<T> {

  public LListCirc2() {
    super();
  }

  public LListCirc2(LList<T> L) {
    super();
    if (!L.isEmpty()) {
      Node<T> p = L.getHead();
      Node<T> res = new Node<T>(p.getData());
      Node<T> w = res;
      while (p.hasNext()) {
        p = p.getNext();
        w.setNext(new Node<T>(p.getData()));
        w = w.getNext();
      }
      w.setNext(res);
      this.setHead(res);
    }
  }

  public void fusion(LListCirc2<T> L) {
    Node<T> p = L.getHead();
    while(p.getNext() != L.getHead()) {
      p = p.getNext();
    }
    p.setNext(this.getHead());
    p = this.getHead();
    while(p.getNext() != this.getHead()) {
      p = p.getNext();
    }
    p.setNext(L.getHead());
  }

  @Override
  public void remove(int index) {
```

```java
    if (index < this.size()) {
      if (index == 0) {
        this.pop(); // remove at index 0
      } else {
        int i = 0;
        Node<T> p = this.getHead();
        while (i < index - 1) {
          i++;
          p = p.getNext();
        }
        Node<T> after = p.getNext().getNext(); // p.next
           exists because index < size
        p.setNext(after);
      }
      this.decSize();
    }
  }

  public static void main(String[] args) {
    // question 1
    LList<Integer> M = new LList<Integer>();
    M.add(1); M.add(2); M.add(3);
    System.out.println(M);

    LListCirc2<Integer> L = new LListCirc2<Integer>(M);
    System.out.println(L);

    // question 2
    LListCirc2<Integer> O = new LListCirc2<Integer>();
    O.add(4); O.add(5); O.add(6);
    System.out.println(O);

    L.fusion(O);
    System.out.println(L);
  }
}
```

java/LListCirc2.java

3 Liste doublement chaînée

On travaille finalement sur les listes doublement chaînées.

1. On suppose disposer d'une liste doublement chaînée dont les éléments de type

`Double` sont triés par ordre croissant. Écrire une méthode `insert(Double elem)` qui permet d'insérer la nouvelle valeur de telle sorte que la liste résultante soit toujours triée par ordre croissant.

2. Écrire une méthode `inverser()` qui produit une liste doublement chaînée dont les éléments sont en ordre inverse de la liste originale.

3. Proposer une méthode `unSurDeux()` qui produit une liste ne contenant qu'un élément sur deux de la liste originale.

Solution :

```
1  package semaine4;
2
3  import linkedList.LList2;
4  import linkedList.NodeDouble;
5
6  public class LList2bis extends LList2<Double>{
7
8    public void insert(Double elem) {
9      if (this.getHead() == null) {
10       this.setHead(new NodeDouble<Double>(elem));
11     } else {
12       NodeDouble<Double> p = this.getHead();
13       if (p.getData() >= elem) this.setHead(new NodeDouble<
            Double>(elem, null, this.getHead()));
14       else {
15         while (p.hasNext() && p.getNext().getData() < elem) {
16           p = p.getNext();
17         }
18         NodeDouble<Double> tmp = new NodeDouble<Double>(elem)
              ;
19         if (p.getNext() != null) {
20           tmp.setPrev(p);
21           tmp.setNext(p.getNext());
22           p.getNext().setPrev(tmp);
23           p.setNext(tmp);
24         } else {
25           // insert in the end
26           p.setNext(tmp);
27           tmp.setPrev(p);
28         }
29       }
30     }
```

```
31    }
32
33    public void inverser() {
34      if (this.getHead() != null) {
35        NodeDouble<Double> p = this.getHead();
36        NodeDouble<Double> res = new NodeDouble<Double>(p.
              getData());
37        while (p.hasNext()) {
38          p = p.getNext();
39          res = new NodeDouble<Double>(p.getData(), null, res);
40        }
41        this.setHead(res);
42      }
43    }
44
45    public void unSurDeux() {
46
47      if (this.getHead() != null) {
48        NodeDouble<Double> p = this.getHead();
49        NodeDouble<Double> res = new NodeDouble<Double>(p.
              getData());
50        NodeDouble<Double> w = res;
51        while (p.hasNext() && p.getNext().hasNext()) {
52          p = p.getNext().getNext();
53          NodeDouble<Double> tmp = new NodeDouble<Double>(p.
                getData());
54          w.setNext(tmp); // set both links!
55          tmp.setPrev(w);
56          w = tmp;
57        }
58        this.setHead(res);
59      }
60    }
61
62    public static void main(String[] args) {
63      // question 1
64      LList2bis L = new LList2bis();
65      L.add(1.0); L.add(2.0); L.add(4.0);
66      System.out.println(L);
67
68      L.insert(3.0);
69      System.out.println(L);
70
71      L.insert(5.0);
```

```
72      System.out.println(L);
73
74      L.insert(0.0);
75      System.out.println(L);
76
77      // question 2
78
79      L.inverser();
80      System.out.println(L);
81
82      // question 3
83      L.unSurDeux();
84      System.out.println(L);
85    }
86  }
```

java/LList2bis.java