TD 7 – Les listes II

Structures de données (IF 122)

Comme la semaine dernière, on s'intéresse à des listes chaînées d'objets de type `Membre`. Un avantage des listes chaînées par rapport aux tableau est que leur taille peut varier. Pour les calculs de complexité, on comptera le nombre d'accès aux champs "`suivant`".

Exercice 1 (TD/TP) Écrire une méthode récursive `longueur` qui retourne le nombre d'éléments d'une liste.

Exercice 2 (TD/TP) Écrire une méthode itérative `getI` qui retourne le membre qui est à la i^{eme} position dans la liste.

Exercice 3 (TD/TP) Donnez cette méthode sous forme récursive (`getR`).

Inverser une liste

Exercice 4 (TD/TP) Écrire une méthode récursive `inverseR` qui inverse l'ordre des éléments d'une liste. Par exemple, l'inverse de (4,2,5,3) est (3,5,2,4). Quelle est la complexité de cette méthode ?

Exercice 5 (TD/TP) Écrire une méthode itérative `inverseI` qui fait la même opération. Quelle est sa complexité ?

Tris

On veut trier les listes de membres par ordre croissant de numéro de membre.

Exercice 6 (TP) Écrire une méthode `insereMembre(Liste l, Membre m)` qui insère un membre `m` au bon endroit dans une liste chaînée `l` qui était déjà triée.

Pour trier toute une liste `l1`, on va utiliser une liste `l2` initialement vide, et on va insérer un a un chacun des éléments de `l1` dans `l2`, ou bon endroit. À la fin, `l2` contiendra tous les éléments, triés.

Exercice 7 (TD/TP) Écrire une méthode `triInsert` qui trie par insertion une liste de membres. Complexité ?

Exercice 8 (TD) On a vu la semaine dernière comment fusionner deux listes triées. On voudrait utiliser cette méthode pour adapter le tri fusion vu sur la feuille 5. Comment peut-on diviser en 2 une liste ?

Exercice 9 (TD/TP) Écrire une méthode qui réalise le tri fusion de la liste de membres.

Listes doublement chaînée

Dans une liste doublement chaînée, chaque cellule contient à la fois un pointeur vers l'élément suivant (`suiv`) et un pointeur vers l'élément précédent (`prec`). Le champ `prec` du premier élément vaut `null`, ainsi que le champ `suiv` du dernier élément. Cela permet de parcourir la liste dans les deux sens.

Exercice 10 (TD/TP) Écrire une méthode qui transforme une liste simplement chaînée en liste doublement chaînée.

Exercice 11 (TD) Comment adapter les méthodes d'insertion et de suppression aux listes doublement chaînées ?

▶ **Exercice 1**

```
import fr.jussieu.script.Deug;

public class Maillon{
    public Membre individu;
    public Maillon suivant;

    /* Exercice 1, feuille 7 */
    public static int nombreElementsRec(Maillon l){
if (l==null)
    return 0;
else
    return 1+nombreElementsRec(l.suivant);
    }

    public static int nombreElementsIter(Maillon l){
int i;
for(i=0;l!=null;i++)
    l = l.suivant;
return i;
    }
```

▶ **Exercice 2**

```
    /* Exercice 2, feuille 7 */
    public static Membre ieme_elementIter(Maillon l,int i){
for(int j=0;j<i;i++)
    { l = l.suivant; }
return l.individu;
    }
```

▶ **Exercice 3**

```
    /* Exercice 3, feuille 7 */
    public static Membre ieme_elementRec(Maillon l,int i){
if (i<=1)
    return l.individu;
else
    return ieme_elementRec(l,i-1);
    }
```

▶ **Exercice 4**

```
    /* Exercice 4, feuille 7 */
    public static Maillon inverseR(Maillon l) {
return inverseAux(l,null);
    }

    public static Maillon inverseAux(Maillon l,Maillon m){
if (l==null)
    return m;
else {
    Maillon aux = l.suivant;
```

```
    l.suivant = m;
    return inverseAux(aux,l);
}
    }
```

▶ **Exercice 5**

```
    /* Exercice 5, feuille 7 */
   public static Maillon inverseI(Maillon l){
Maillon m=null;
Maillon aux;
while (l!=null){
    aux = l.suivant;
    l.suivant = m;
    m=l;
    l=aux;
}
return m;
   }

    /* extraits de la feuille 6 */
    public static void affiche_liste_static(Maillon m){
while(m!=null){
    m.individu.affiche();
    m = m.suivant;
}
Deug.println();
    }

    public static Membre cherche_membre(Maillon m,int i){
while(m!=null){
    if (m.individu.getNumero()==i) return m.individu;
    m = m.suivant;
}
return null;
    }

    public static Maillon enleve_membre(Maillon ma,int i){
if (ma.individu.getNumero()==i)
    return ma.suivant;
Maillon mb=ma;
while(mb.suivant.individu.getNumero()!=i) {
    mb=mb.suivant;
}
mb.suivant=mb.suivant.suivant;
return ma;
    }
}

class TestM {
    public static void main(String[] args) {

Maillon m1= new Maillon();
m1.individu = new Membre(12,"Bernard","Alain");

Maillon m2= new Maillon();
```

```
m1.suivant= m2;
m2.individu = new Membre(16,"Delle","Claude");

Maillon m3= new Maillon();
m2.suivant= m3;
m3.individu = new Membre(18,"Fabre","Etienne");

Maillon.affiche_liste_static(m1);

m1=Maillon.inverseR(m1);
Maillon.affiche_liste_static(m1);

m1=Maillon.inverseI(m1);
Maillon.affiche_liste_static(m1);

/*
Membre lui; // lui= new Membre(12,"Bernard","Alain");
// lui.affiche();

int i=14;
lui= Maillon.cherche_membre(m1,i);
if (lui==null) Deug.println("pas trouvÃ© le membre numÃ©ro "+i);
else {
    Deug.print("trouvÃ© le membre numÃ©ro "+i+" : ");
    lui.affiche();
}
Deug.println();

m1=Maillon.enleve_membre(m1,12);
Maillon.affiche_liste_static(m1);
Deug.println();

m1=Maillon.enleve_membre(m1,16);
Maillon.affiche_liste_static(m1);
Deug.println();

// m1.individu.affiche();
// m2.individu.affiche();
*/
    }
}
```