NFA032 : TP 13
Structures et méthodes récursives

11 juin 2018

Exercice 1 Listes d'entiers

Nous allons définir une structure de donnée récursive permettant de représenter une liste d'entiers. Chaque objet de cette structure contient un nombre entier et une variable contenant l'adresse de l'objet suivant, qui contient l'entier suivant dans la liste.

```
public class ListeEntier {
    private int valeur;
    private ListeEntier suivant;
    public ListeEntier(int val, ListeEntier suiv) {
        this.valeur = val;
        this.suivant = suiv;
    }
    public ListeEntier(int val) {
        this.valeur = val;
        this.suivant = null;
    }
    public int getValeur() {
        return valeur;
    }
    public void setValeur(int val) {
        this.valeur = val;
    }
    public ListeEntier getSuivant() {
        return suivant;
    }
    public void setSuivant(ListeEntier suiv) {
        this.suivant = suiv;
    }
}
```

Les constructeurs `ListeEntier` permettent d'ajouter un entier en début de liste et renvoient un objet représentant ce premier élément de liste.
Voici un bout de programme.

```
ListeEntier lal = new ListeEntier(4);
lal = new ListeEntier(10,lal);
lal = new ListeEntier(7,lal);
lal = new ListeEntier(3,lal);
```

1. Manipulation d'une liste
   (a) dessinez la liste `lal`.
   (b) par quel code Java utilisant la liste `lal` peut-on désigner l'élément 3 de la liste ?
   (c) par quel code Java utilisant la liste `lal` peut-on désigner l'élément 7 de la liste ?

(d) par quel code Java utilisant la liste `lal` peut-on désigner l'élément 4 de la liste ?

(e) par quel code Java utilisant la liste `lal` désigner la portion de liste qui ne contient que les deux derniers éléments de la liste ?

(f) par quel code java peut-on remplacer l'entier 3 par 5, dans la liste ?

2. Ajout en fin de liste

   (a) définissez une nouvelle méthode `ajouterALaFin` dans la classe `ListeEntier` qui ajoute un entier à la fin de la liste (dans un nouvel objet de type `ListeEntier`.

   (b) construisez la même liste que `lal` (mêmes éléments dans le même ordre) avec un appel à un constructeur suivi de trois appels à la méthode `ajouterALaFin`.

3. Parcours de la liste

   (a) Écrire la méthode `toString()` associée à cette classe :
   — de façon itérative
   — puis de façon récursive

   (b) Écrire une méthode `taille()` calculant le nombred'éléments d'une liste :
   — de façon itérative
   — puis de façon récursive

Exercice 2 Insertion d'élément

Le but de cet exercice est de vous familiariser avec les listes et avec la notation graphique associée. Cette notation n'est qu'une représentation abstraite de ce qui se passe en mémoire. Elle peut nous aider, nous humains, à comprendre ce qui se passe en machine. Cela peut nous aider à raisonner pour faire des programmes corrects.

1. Notre but est d'écrire une fonction qui ajoute un élément à un rang donné d'une liste. Nous avons déjà vu l'insertion en début et en fin de liste. Avec l'insertion à un rang donné, nous complétons la panoplie de méthodes permettant d'ajouter un élément à une liste. Procédons par étape.

   L'algorithme consiste à avancer dans la liste jusqu'au bon endroit. Là, il faut insérer la nouvelle cellule. Il faut que la cellule précédente pointe sur elle et qu'elle pointe sur l'élément suivant.

   Voyons d'abord un exemple.

   Supposons que la liste contient 1, 2, 3, 4 et qu'on veut insérer 5 en position 3 (comme troisième élément de la liste). Dessinez la représentation de l'état initial. Indice : il faut une variable auxiliaire de type ListeEntier pour parcourir la liste et éventuellement une seconde variable du même type pour contenir le nouvel objet qui contiendra la valeur 5.

   Dessinez ensuite tous les états intermédiaires par lesquels il faut passer pour réaliser l'opération. Faites bien un dessin séparé pour chaque état.

   Essayez de décrire le changement d'état par une instruction Java (ou une portion d'instruction comme nous l'avons fait dans les rappels sur les listes). Cette instruction doit être une affectation ou un appel de méthode du genre :

   — `lis1 = lis2;`
   — `lis=lis.getSuivant();`
   — `lis1= new ElementListe(x,lis2);`
   — `lis1.setSuivant(l2);`

Si vous avez besoin de plus d'une instruction pour décrire le passage d'un état au suivant, c'est que vous avez oublié un état intermédiaire entre les deux.

S'il n'est pas possible de trouver une instruction qui décrive le passage d'un état à un autre, c'est que vous vous êtes permis de faire des choses impossibles à réaliser en Java.

2. Arrivé ici, vous devez avoir votre séquence de dessins et en parallèle la séquence d'instructions correspondante. Si les choses ne sont pas encore bien claires, refaites la même chose pour d'autres exemples de listes, d'entier à ajouter et de rang.
   Ecrivez une procédure qui fonctionne dans tous les cas simples (comme l'exemple étudié). Dans cette première version, on traitera pas les cas particuliers ni les problèmes qui peuvent arriver. Testez votre procédure.
   En gros, on doit retrouver les différentes affectations de l'exemple, organisées avec des structures de contrôle (boucles, if). Eventuellement, il faudra de nouvelles variables pour décrire les conditions de ces structures de contrôle.
3. Nous allons maintenant parachever le travail en traitant les erreurs et les cas particuliers éventuels.

   Cas particuliers :
   — la procédure marche-t-elle si on veut ajouter l'élément en première position ?
   — la procédure marche-t-elle si on veut ajouter l'élément en dernière position ?
   Modifiez la procédure pour qu'elle marche dans tous ces cas.

   Le problème qui peut arriver est que la liste ne comporte pas assez d'éléments pour que le rang demandé existe. Par exemple, il n'est pas possible d'insérer au dixième rang d'une liste de trois entiers.

   A quel endroit de la procédure peut-on déceler le problème, et comment ?