TP 6– Listes doublement chaînées et algorithmes associés

Année 2019-2020

Dans le cadre de ce TP, on s'intéresse à la mise en oeuvre de la classe liste par une liste doublement chaînée. Chaque élément de la liste comporte une valeur (valeur de l'élément), une référence sur l'élément suivant et une référence sur l'élément précédent. Tous les exercices proposés dans ce TP utilisent cette représentation de liste (ou une variante pour l'exercice 3).

Exercice 1 – Implémentation d'une liste doublement chaînée

L'objectif de cet exercice et de proposer une mise en oeuvre de la classe `DoubleLinkedList` donnée en annexe (fichier .py sur Moodle) à partir d'une liste doublement chaînée. Chaque élément de la liste est une cellule (class Cellule) définie par la structure suivante :

```
class Cellule():
    def __init__(self):
        self.value = object
        self.next = None
        self.prev = None
```

Chaque cellule porte une valeur (`value` de type object), et pointe sur la cellule suivante grâce à son champ `next` et vers la cellule précédente grâce à son champ `prev`. Le nombre de cellules dans la liste sera égal au nombre de valeurs à stocker. La dernière cellule n'a pas de suivant.

1. Spécifiez les méthodes de la classe `DoubleLinkedList` fournie en annexe.
2. Implémentez l'ensemble des méthodes définies dans la partie spécification de la classe `DoubleLinkedList`.
3. Afin de tester au fur et à mesure les méthodes de la classe `DoubleLinkedList`, vous écrirez une classe `TestList` qui permet de créer une liste d'objets et de tester les différentes méthodes d'insertion et de suppression. Par exemple, vous pourrez reprendre comme objets ceux représentés par la classe `Bowl` qui modélise une boule de couleur.

Exercice 2 – Tri dans une liste doublement chaînée

On s'intéresse dans cet exercice à l'utilisation d'une liste doublement chaînée.

1. Définissez une classe `Bowl` qui modélise une boule de couleur ; vous implémenterez une méthode `compare(b)` permettant de comparer les boules entre elles.
2. Définissez une classe utilisateur qui permet de construire une liste doublement chaînée constituée de boules dont la couleur est tirée de manière aléatoire.

3. Ecrivez un algorithme qui permet de trier ces boules. Vous prendrez l'algorithme de tri par permutations (tri bulle) qui permet de remonter par permutations de proche en proche la valeur maximale du tableau, puis de redescendre par permutations la valeur minimale du tableau en modifiant à chaque passage les indices de début et de fin du tableau.

Exercice 3 – AmStramGram

On s'intéresse dans cet exercice à la mise en oeuvre de la classe liste par une liste circulaire doublement chaînée, dans laquelle le dernier élément de la liste est connecté au premier. Vous pourrez reprendre l'implémentation de la liste doublement chaînée en apportant les modifications nécessaires.

1. Écrivez la classe générique `CircularList` qui implémente la classe `DoubleLinkedList`. Vous définirez les références sur le premier élément et le dernier (`head` et `last`), ainsi que le constructeur `CircularList()`.

2. Modifiez les méthodes qui permettent d'insérer des éléments et d'en supprimer.

3. Pour jouer à $AM - STRAM - GRAM$, n enfants forment une ronde, choisissent l'un d'entre eux comme le premier, commencent à partir de lui et décident que le k-ème enfant doit quitter la ronde, puis le 2.k-ème, et ainsi de suite. En représentant la ronde des enfants par une liste circulaire doublement chaînée, écrivez un programme qui donne la suite des enfants (par leur nom par exemple) dans l'ordre où ils sont sortis de la ronde.

Spécification de la classe doublement chaînée DoubleLinkedList
Spécification à compléter

```
class DoubleLinkedList:
    """ Liste doublement chainee """

    def __init__(self):
        """ """
    def __str__(self):
        """ """
    def sizeList(self):
        """ donne la taille de la liste"""
    def emptyList(self):
        """ verifie si la liste est vide ou non """
    def insertFirst(self, v):
        """ insere l'element de valeur v en premiere position"""
    def insertLast(self, v):
        """insere l'element de valeur v en derniere position"""
    def insertBefore(self, v, elt):
        """insere la valeur v avant l'element elt de type Cellule """
    def insertAfter(self, v, elt):
        """ insere la valeur v apres l'element elt de type Cellule """
    def insertAt(self, v, n):
        """ insere l'element de valeur v en nieme position """
    def head(self):
        """retourne la valeur du premier element de la liste"""
    def getFirst(self):
```

```
        """ retourne le premier element """
    def getLast(self):
        """ retourne le dernier element """
    def getAt(self, n):
         """ retourne la valeur du nieme element """
    def getEltAt(self, n):
        """ recherche le nieme element de la liste (d'indice n-1) """
    def remove(self, v):
        """ retire la premiere occurrence de l'element de valeur v """
    def removeAt(self, n):
        """ retire le nieme element de la liste (d'indice n-1) """
    def removeFirst(self):
        """ retourne la liste dans laquelle on a retire le premier element """

class Cellule():
```