TD_Listes

May 29, 2020

```
In [242]: import sys
```

1 Classe liste

Construire une classe implémentant les listes doublement chaînées non circulaires incluant :

- un constructeur et un destructeur
- deux attributs premier et dernier qui sont des cellules
- des méthodes estVide, vider, nbElements, iemeElement, modifierIemeElement, afficher, ajouterEnTete, ajouterEnQueue, supprimerTete, rechercherElement et insererElement

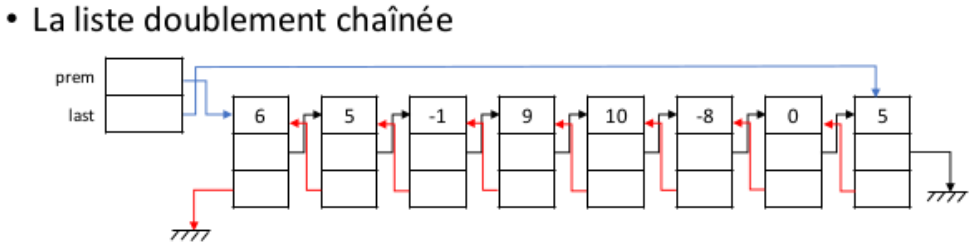
Vous définirez pour cela une classe-structure Cellule composée de trois attributs : info, suivant et precedent.

Écrire une procédure de la classe Liste qui, à partir d'un tableau d'éléments (liste Python), crée la liste chaînée contenant les mêmes éléments dans le même ordre. Donnez un exemple d'appel à cette procédure.

Écrivre une procédure de la classe Liste qui trie les éléments par ordre croissant, en utilisant l'algorithme du tri par insertion.

```
In [243]: class  Liste:

              def __init__(self):
                  self.premier = None
                  self.dernier = None
                  self.taille = 0
```



Liste doublement chainée non circulaire

- Fonction iemeElement (indice) : tout type
  - Précondition : 0 <= indice < nombre d'éléments
  - Résultat : retourne l'élément à l'indice passé en paramètre
- Procédure modifierIemeElement (e, indice)
  - Précondition : 0 <= indice < nombre d'éléments
  - Postcondition : l'élément à l'indice passé en paramètre vaut e
- Procédure afficher ()
  - Postcondition : Les éléments de la liste sont affichés à l'écran
- Procédure supprimerTete ()
  - Postcondition : l'élément en tête de liste est supprimé
- Procédure insererElement (e, indice)
  - Précondition : 0 <= indice <= nombre d'éléments
  - Postcondition : e est inséré de sorte qu'il occupe la position d'indice en paramètre
- Fonction rechercheElement (e) : entier
  - Résultat : retourne l'indice de l'élément e dans la liste, ou -1 si l'élément n'est pas présent

Fonctionnalités listes 1

- Constructeur Liste()
  - Postconditions : la liste est une liste vide
- Destructeur ~Liste()
  - Postconditions : libération de la mémoire utilisée, la liste est une liste vide
- Procédure ajouterEnTete (e)
  - Postcondition : l'élément e est ajouté en tête de liste
- Procédure ajouterEnQueue (e)
  - Postcondition : l'élément e est ajouté en queue de liste
- Procédure vider ()
  - Postcondition : la liste ne contient plus aucune cellule
- Fonction estVide () : booléen
  - Résultat : retourne vrai si la liste est vide, faux sinon

Fonctionnalités listes 2

```
def est_vide(self):
    #return self.premier is None and self.dernier is None
    return self.premier is None

def vider(self):
    while self.premier is not None:
        poubelle = self.premier
        nouveau = self.premier.suivant
        if nouveau is not None:
            nouveau.precedent = self.premier
        self.premier = nouveau
        del poubelle
        self.taille -= 1
    self.dernier = self.premier


def detruire(self):
    self.vider()
    del self.dernier
    del self.premier
    del self.taille


def ajouterEnTete(self, e):
    nouveau = Cellule(e, None, self.premier)
    if self.est_vide():  #si la liste est vide
        self.dernier = self.premier = nouveau
    else:
        self.premier.precedent = nouveau
        self.premier = nouveau
    self.taille += 1


def ajouterEnQueue(self, e):
    nouveau = Cellule(e, self.dernier, None)
    if self.est_vide():  #si la liste est vide
        self.dernier = self.premier = nouveau
    else:
        self.dernier.suivant = nouveau
        self.dernier = nouveau
    self.taille += 1

def afficher(self):
    pointeur = self.premier
    output = "["
    while pointeur is not None:
        output += str(pointeur.info) + ','
        pointeur = pointeur.suivant
```

```
        print(output.rstrip(',') + ']')

    def __str__(self):
        pointeur = self.premier
        output = "["
        while pointeur is not None:
            output += str(pointeur.info) + ','
            pointeur = pointeur.suivant
        return output.rstrip(',') + ']'


    def nb_elements(self):
        #return self.taille
        pointeur = self.premier
        compteur = 0
        while pointeur is not None:
            pointeur = pointeur.suivant
            compteur += 1
        return compteur

    def iemeElement(self, i):
        assert 0 <= i < self.nb_elements()
        index = 0
        pointeur = self.premier
        while pointeur is not None and index != i:
            index += 1
            pointeur = pointeur.suivant
        if pointeur is not None:
            return pointeur.info
        else:
            return None

    def modifierIemeElement(self, e, i):
        assert 0 <= i < self.nb_elements()
        index = 0
        pointeur = self.premier
        while pointeur is not None and index != i:
            index += 1
            pointeur = pointeur.suivant
        if pointeur is not None:
            pointeur.info = e


    def supprimerTete(self):
        if not self.est_vide():
            poubelle = self.premier
            nouveau = self.premier.suivant
            if nouveau is not None:
```

```
            nouveau.precedent = self.premier
        self.premier = nouveau
        if self.est_vide():
            self.dernier = self.premier
        del poubelle
    self.taille -= 1

def rechercherElement(self, e):
    pointeur = self.premier
    index = 0
    while pointeur is not None and pointeur.info != e:
        pointeur = pointeur.suivant
        index += 1
    return -1 if pointeur is None else index

def insererElement(self, e, i):
    n = self.nb_elements()
    assert 0 <= i <= n
    if i == 0:
        self.ajouterEnTete(e)
    elif i == n:
        self.ajouterEnQueue(e)
    elif i < n // 2:
        pointeur = self.premier
        index = 0
        while pointeur is not None and index != i:
            pointeur = pointeur.suivant
            index += 1
        avant = pointeur.precedent
        apres = pointeur
        nouveau = Cellule(e, avant, apres)
        avant.suivant = nouveau
        apres.precedent = nouveau
    else:
        pointeur = self.dernier
        index = n - 1
        while pointeur is not None and index != i:
            pointeur = pointeur.precedent
            index -= 1
        avant = pointeur.precedent
        apres = pointeur
        nouveau = Cellule(e, avant, apres)
        avant.suivant = nouveau
        apres.precedent = nouveau
    self.taille += 1

@staticmethod
def doubleChainedList_from_Pythonlist(t):
```

```
            L = Liste()
            for e in t:
                L.ajouterEnQueue(e)
            return L

        def triInsertion(self):
            element = self.premier
            while element is not None:
                avant = element.precedent
                val = element.info
                while avant is not None and avant.info > val:
                    avant.suivant.info = avant.info
                    avant = avant.precedent
                if avant is None:
                    self.premier.info = val
                else:
                    avant.suivant.info = val
                element = element.suivant


    class Cellule:

        def __init__(self, info, precedent, suivant):
            self.info = info
            self.precedent = precedent
            self.suivant = suivant
```

In [244]:
```
L = Liste()
print(f"Attributs : {vars(L)}, identifiants : {id(L)} et taille en octets : {sys.gets


print("-----------------------")


print("Ajout de list(range(6)) avec L.ajouterEnTete()")
for k in range(6):
    L.ajouterEnTete(k)
print("Affichage de la liste")
L.afficher()
n = L.nb_elements()
print(f"Affichage du nombre d'éléments : {n}")
for k in range(6):
    print(f" 0 <= k < L.nb_elements() : {0 <= k < n} et {k}-ieme élément : {L.iemeEle
print(f"Premier : {L.premier.info} et Dernier : {L.dernier.info} ")

print("-----------------------")
```

```
print("On vide la liste")
L.vider()
print("Affichage de la liste")
L.afficher()
print(f"attributs : {vars(L)}, identifiants : {id(L)} et taille en octets : {sys.gets


print("-----------------------")


print("On détruit la liste")
L.detruire()
print(f"attributs : {vars(L)}, identifiants : {id(L)} et taille en octets : {sys.gets


print("-----------------------")
L = Liste()
print("Ajout de list(range(6)) avec L.ajouterEnQueue()")
for k in range(6):
    L.ajouterEnQueue(k)
print("Affichage de la liste")
L.afficher()
print("Affichage de la liste")
L.afficher()
for k in range(6):
    print(f" 0 <= k < L.nb_elements() : {0 <= k < n}, on remplace le  {k}-ieme éléme
          f" par {k+1}  ")
    L.modifierIemeElement(k + 1, k)
print("Affichage de la liste")
L.afficher()
print(f"Premier : {L.premier.info} et Dernier : {L.dernier.info} ")

print("-----------------------")
print("Affichage de la liste")
L.afficher()
print("Vidage de la liste avec supprimerTete")
while L.premier is not None:
    L.supprimerTete()
print("Affichage de la liste")
L.afficher()
print(f"attributs : {vars(L)}, identifiants : {id(L)} et taille en octets : {sys.gets


print("-----------------------")
print("Ajout de list(range(6)) avec L.ajouterEnTete()")
for k in range(6):
```

```
            L.ajouterEnTete(k)
        print("Affichage de la liste")
        L.afficher()
        print(f"Insertion d'un élément en position {L.nb_elements() - 1},  dans la seconde mo
        L.insererElement(5.5, L.nb_elements() - 1)
        print("Affichage de la liste")
        L.afficher()
        print(f"Insertion d'un élément en position {1},  dans la première moitié")
        L.insererElement(0.5, 1)
        print("Affichage de la liste")
        L.afficher()
```

```
Attributs : {'premier': None, 'dernier': None, 'taille': 0}, identifiants : 139820595078928 et
-----------------------
Ajout de list(range(6)) avec L.ajouterEnTete()
Affichage de la liste
[5,4,3,2,1,0]
Affichage du nombre d'éléments : 6
 0 <= k < L.nb_elements() : True et 0-ieme élément : 5
 0 <= k < L.nb_elements() : True et 1-ieme élément : 4
 0 <= k < L.nb_elements() : True et 2-ieme élément : 3
 0 <= k < L.nb_elements() : True et 3-ieme élément : 2
 0 <= k < L.nb_elements() : True et 4-ieme élément : 1
 0 <= k < L.nb_elements() : True et 5-ieme élément : 0
Premier : 5 et Dernier : 0
-----------------------
On vide la liste
Affichage de la liste
[]
attributs : {'premier': None, 'dernier': None, 'taille': 0}, identifiants : 139820595078928 et
-----------------------
On détruit la liste
attributs : {}, identifiants : 139820595078928 et taille en octets : 56
-----------------------
Ajout de list(range(6)) avec L.ajouterEnQueue()
Affichage de la liste
[0,1,2,3,4,5]
Affichage de la liste
[0,1,2,3,4,5]
 0 <= k < L.nb_elements() : True, on remplace le  0-ieme élément : 0 par 1
 0 <= k < L.nb_elements() : True, on remplace le  1-ieme élément : 1 par 2
 0 <= k < L.nb_elements() : True, on remplace le  2-ieme élément : 2 par 3
 0 <= k < L.nb_elements() : True, on remplace le  3-ieme élément : 3 par 4
 0 <= k < L.nb_elements() : True, on remplace le  4-ieme élément : 4 par 5
 0 <= k < L.nb_elements() : True, on remplace le  5-ieme élément : 5 par 6
Affichage de la liste
[1,2,3,4,5,6]
Premier : 1 et Dernier : 6
```

```
----------------------
Affichage de la liste
[1,2,3,4,5,6]
Vidage de la liste avec supprimerTete
Affichage de la liste
[]
attributs : {'premier': None, 'dernier': None, 'taille': 0}, identifiants : 139820595080104 et
----------------------
Ajout de list(range(6)) avec L.ajouterEnTete()
Affichage de la liste
[5,4,3,2,1,0]
Insertion d'un élément en position 5,  dans la seconde moitié
Affichage de la liste
[5,4,3,2,1,5.5,0]
Insertion d'un élément en position 1,  dans la première moitié
Affichage de la liste
[5,0.5,4,3,2,1,5.5,0]
```

1.1 Création d'une liste chaînée à partir d'un tableau

Écrire une procédure de la classe Liste qui, à partir d'un tableau d'éléments (liste Python), crée la liste chaînée contenant les mêmes éléments dans le même ordre. Donnez un exemple d'appel à cette procédure.

```
In [245]: L = Liste.doubleChainedList_from_Pythonlist([1,2,3])

In [246]: L.afficher()

[1,2,3]


In [247]: from random import randint

In [248]: L = Liste.doubleChainedList_from_Pythonlist([randint(1, 100) for _ in range(10)])
          print("Avant tri ", L)
          L.triInsertion()
          print("Après tri ",L)

Avant tri  [17,16,9,81,58,8,18,36,65,68]
Après tri  [8,9,16,17,18,36,58,65,68,81]


In [249]: #Test du tri par insertion sur les permutations de {0,1,2,3,4,5}

          from itertools import permutations

          collector = []
          base = '[' + ','.join(map(str, list(range(6)))) + ']'
```

```
        for p in permutations(range(6)):
            t = Liste.doubleChainedList_from_Pythonlist(list(p))
            t.triInsertion()
            collector.append(str(t) == base)
        print(all(collector))
```

```
True
```