Numérique et science informatique
Classe de Terminale

Lycée hoche

année scolaire 2021-2022

Contents

Récursivité

On se limite aux types abstraits de données qui stockent les données sous forme de collection unidimensionnelle ; ce sont les types linéaires, dont nous présentons les listes, les piles et les files.

1 Liste

Définition

Une liste est une collection finie de données.Elle permet de stocker des données et d'y accéder directement.C'est un **type abstrait de données**qui est:

- linéaire:Les données sont stockées dans une structure unidimensionnelle.
- indexé: Chaque donnée est associée à une valeur.
- ordonné: Les données sont présentées les unes après les autres.

On appelle "tête" le premier élément de la liste et "queue" la liste privée de son premier élément.Il est seulement possible d'ajouter et de lire une donnée en tête de liste .
Cinq primitives permettent de définir le type abstrait de données:

- listeCree() : crée la liste vide.
- listeAjout() :ajoute un élément en tête de liste.
- listeTete() :renvoie la valeur de l'élément en tête de liste.
- listeQueue() :renvoie la liste privée de son premier élément.
- listeEstVide() :renvoie vrai si la liste est vide ,faux sinon.

Remarque : Le type abstrait de données est **non mutable**.

2 Pile

2.1 présentation

La pile, comme la liste, permet de stocker des données et d'y accéder. La différence se situe au niveau de l'ajout et du retrait d'éléments.On parle de mode LIFO (Last In, First Out, donc, dernier arrivé, premier sorti), c'est-à-dire que le dernier élément ajouté à la structure sera le prochain élément auquel on accèdera. Les premiers éléments ayant été ajoutés devront « attendre » que tous les éléments qui ont été ajoutés après eux soient sortis de la pile. Contrairement aux listes, on ne peut donc pas accéder à n'importe quelle valeur de la structure (pas d'index). Pour gérer cette contrainte, on définit alors le sommet de la pile qui caractérise l'emplacement pour ajouter ou retirer des éléments.On peut s'imaginer une pile d'assiettes pour mieux se représenter mentalement cette structure. On ajoute des nouvelles assiettes au sommet de la pile, et quand on veut en retirer une, on est obligé de prendre celle située au sommet.

Définition

Une Pile est une collection finie de données.
Le **sommet** de la pile est le dernier élément ajouté qui est aussi le seul élément auquel on peut accéder, ou qu'on peut retirer.Quand un élément est ajouté à la pile, il devient le nouveau sommet, et l'ancien sommet est alors son élément **suivant**.Quand un élément est retiré de la pile ,le nouveau sommet est l'élément qui suivait le sommet avant le retrait.

Six primitives permettent de définir le type abstrait de données:

- pileCree() : crée une pile vide.
- pileTaille(P) : renvoie le nombre d'éléments contenus dans la pile.
- PileEstVide(P) :renvoie vrai si la pile est vide , faux sinon.
- PileEmpiler(P) :ajoute un élément au sommet de la pile (qui devient le nouveau sommet)
- Piledepiler(P) :retire et renvoie le sommet de la pile.
- PileSommet(P) :renvoie l'élément qui se trouve au sommet de la pile sans le retirer.

2.2 Remarques

La pile est utile dans différents types de problèmes:

- algorithme d'un navigateur pour pouvoir mémoriser les pages web et revenir en arrière (ou réavancer) sur certaines pages.
- stocker des actions et les annuler (ou les réappliquer) , sur l'ordinateur (CTRL+Z et CTRL+Y).
- Coder une calculatrice en notation polonaise inversée.
- algorithme du parcours en profondeur pour les arbres et les graghes par exemple,pour résoudre un labyrinthe par exemple,trouver un trajet sur une carte.
- Ecrire des versions itératives de certains algorithmes récursifs.
- illustration du fonctionnement de la pile d'appels des fonctions lors de l'exécution d'un programme.

3 File

3.1 Présentation

La file, comme la liste, permet de stocker des données et d'y accéder. La différence se situe au niveau de l'ajout et du retrait d'éléments.
On parle de mode FIFO (First in, First out, donc, premier arrivé, premier sorti), c'est-à-dire que le premier élément ayant été ajouté à la structure sera le prochain élément auquel on accèdera.

Les derniers éléments ajoutés devront « attendre » que tous les éléments ayant été ajoutés avant eux soient sortis de la file. Contrairement aux listes, on ne peut donc pas accéder à n'importe quelle valeur de la structure (pas d'index).
Pour gérer cette contrainte, la pile est caractérisée par deux « emplacements » :

- la tête de file, sortie de la file (début de la structure), où les éléments sont retirés ;
- le bout de file, entrée de la file (fin de la structure), où les éléments sont ajoutés.

On peut s'imaginer une file d'attente, dans un cinéma par exemple. Les premières personnes à pouvoir acheter leur place sont les premières arrivées, et les nouveaux arrivants se placent au bout de la file.

Définition

Une file est une collection de données.On appelle **tête de file** le premier élément de la structure et **bout de file** le dernier élément.Quand un élément est ajouté à la fin de la file, on l'ajoute en bout de file et il devient le nouveau bout de file, c'est-à-dire l'élément suivant l'élément précédemment situé en bout de file.

Quand un élément est retiré de la file,on le sélectionne à la tête de la file et la nouvelle tête est l'élément qui suivait l'ancienne tête.Lorsqu'on ajoute un élément à une file vide,celui-ci est donc à la fois tête et bout de file.

Six primitives permettent de définir le type abstrait de données:

- fileCree() : crée une pile vide.
- fileTaille(F) : renvoie le nombre d'éléments contenus dans la file.
- fileEstVide(F) :renvoie vrai si la file est vide , faux sinon.
- fileAjouterFin(F) :ajoute un élément au bout de la file (qui devient le nouveau bout de fil)
- FileretirerTete(F) :retire et renvoie à la tête de la file.
- fileTete(P) :renvoie l'élément qui se trouve à la tête de la file (sans le retirer).

3.2 Remarques

La file est utile dans différents types de problèmes :
• pour une imprimante, gestion de la file d'attente des documents à imprimer ;

• modélisation du jeu de la bataille (on révèle la carte au-dessus du paquet et on place celles gagnées en dessous...) ;

• gestion de mémoires tampon, pour gérer les flux de lecture et d'écriture dans un fichier, par exemple ;
• matérialisation d'une file d'attente, pour un logiciel (visioconférence par exemple) ou un jeu (gestion des connexions des utilisateurs, des tours de jeu...),...
• algorithme du parcours en largeur pour les arbres et les graphes, par exemple, pour trouver le plus court trajet sur une carte, ou récupérer les valeurs d'une structure

4 Implantation de structures de données

4.1 Implantation du type abstrait de données liste avec les listes Python

```
def listeCree() :
    '''
    crée une liste vide en s appuyant sur les listes Python
    '''
    return []


def listeAjout(liste, element) :
    ''' Paramètres
    liste : une liste à laquelle on souhaite ajouter un élément
    element : lélément à ajouter, de type quelconque
    '''

    # ajoute un élément en tête de liste
    liste.append(element)


def listeTete(liste) :
    ''' Paramètres
    liste : une liste
    '''
    #renvoie la valeur de lélément en tête de liste
    if not listeEstVide(liste) :
        return liste[-1]
    else :
        return None


def listeQueue(liste) :
    ''' Paramètres
    liste : une liste    :   renvoie la queue de la liste
    '''
    return liste[:-1]
    # il peut être pertinent d utiliser la syntaxe liste.pop() afin d é
   viter le slicing

def listeEstVide(liste) :
    ''' Paramètres
    liste : une liste        renvoie ``True`` si la liste est vide, ``
   False`` sinon
    '''
    return len(liste)==0
```

4.2 Implantation du type abstrait de données 'pile' avec les listes Python

Une mise en œuvre de la structure réalisant le type abstrait de données pile de la ressource « Types abstraits de données - Présentation » peut être proposée dès le début de la terminale, en n'utilisant que les acquis de première : Implantation du type Abstrait de données pile avec les listes Python

```
def pileCree() :

    '''    Crée une pile vide en s'appuyant sur les listes Python
    '''
    return []


def pileTaille(pile):


    '''    Paramètres :   pile : Une pile, telle que créée dans ce
   module (utilisant une list Python)
    Description : La pile dont on veut connaître le nombre d'éléments
    :   Renvoie le nombre d'éléments contenus dans la pile.
    '''
    return len(pile)

def pileEstVide(pile) :

    '''    Paramètres      pile : Une pile, telle que créée dans ce
   module (utilisant une list Python)
    Description : La pile dont on souhaite déterminer si elle est vide.
    Renvoie ``True`` si la pile est vide, et ``False`` sinon
    '''
    return pileTaille(pile) == 0


def pileEmpiler(pile, element) :

    '''
    Paramètres  -    pile : Une pile, telle que créée dans ce module (
   utilisant une list Python)        Description : La pile sur laquelle
   on souhaite empiler un élément    element : N'importe quel type de
   données.
    Description : L'élément à empiler dans la pile.       Empile un élé
   ment dans la pile passée en paramètres.
    '''
    pile.append(element)



def pileDepiler(pile) :
    '''
    Paramètres       pile : Une pile, telle que créée dans ce module (
   utilisant une list Python)        Description : La pile sur laquelle
   on souhaite dépiler un élément
    Dépile (supprime de la pile) l'élément au sommet de la pile et le
   renvoie.
    '''
    if pileEstVide(pile) :
        return None
    else :
        return pile.pop()
```

```
def pileSommet(pile):
    '''
    Paramètres        pile : Une pile, telle que créée dans ce module (
   utilisant une list Python)
    Description : La pile dont on veut connaître le sommet.
   Renvoie le sommet de la pile (mais ne le dépile pas).
    '''
    if pileEstVide(pile) :
        return None
    else :
        return pile[len(pile) - 1]
```

4.3 Implantation du type abstrait de données 'file' avec les listes Python

Une mise en œuvre de la structure réalisant le type abstrait de données file de la ressource « Types abstraits de données - Présentation » peut être proposée dès le début de la terminale, en n'utilisant que les acquis de première :

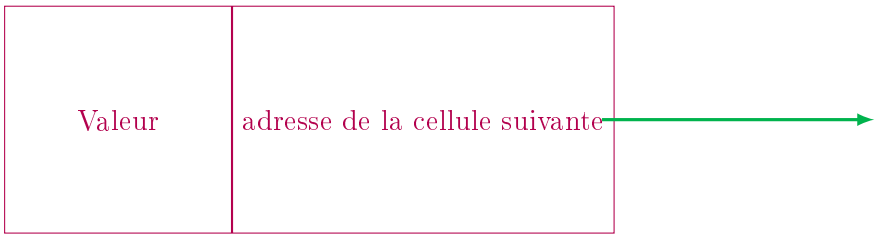
```
# Implantation du type Abstrait de données file avec les listes Python.



def fileCree() :
    '''
    Crée une file vide en s'appuyant sur les listes Python.
    '''

    return []

def fileTaille(file) :
    '''   Paramètres        file : Une file, telle que créée dans ce
   module (utilisant donc une listes Python)        Description : La
   file dont on veut connaître le nombre d'éléments    ———————
   Renvoie le nombre d'éléments contenus dans la file

    '''
    return len(file)


def fileVide(file) :

    '''
    Paramètres      file : Une file, telle que créée dans ce module (
   utilisant donc une liste Python)
    Description : La file qu'on veut vérifier.    Renvoie 'True' si la
   file est vide, et 'False' sinon
    '''

    return fileTaille(file) == 0


def fileAjouterFin(file, element) :
```

```
    '''    Paramètres       file : Une file, telle que créée dans ce
    module (utilisant donc une liste Python)

    Description : La file à laquelle on souhaite ajouter un élément
    element : N'importe quel type de données.       Description : L'élé
    ment à ajouter au bout de la file.      Ajoute un élément au bout de
    la file passée en paramètres.
    '''
    file.append(element)


def fileRetirerTete(file) :
'''   Paramètres     file : Une file, telle que créée dans ce module (
    utilisant donc une liste Python)       Description : La file à
    laquelle on souhaite retirer un élément      Retirer (supprime de la
    file) et renvoie l'élément en tête (situé au début)     de la file.
'''

if fileVide(file) :
    return None
else :
    return file.pop(0)


def fileTete(file) :
    '''    Paramètres      file : Une file, telle que créée dans ce
    module (utilisant donc une liste Python)       Description : La file
    dont on veut connaître la tête.     Renvoie l'élément en tête (situé
    au début) de la file (mais ne le supprime pas)
    '''

    if fileVide(file) :
        return None
    else :
        return file[0]
```

5 Implémentation utilisant la programmation orientée objet

Que cela soit pour les listes, les piles ou les files, ces structures sont définies par un ensemble d'éléments ordonnés et reliés entre eux. Pour faciliter les implantations de ces structures en utilisant la programmation orientée objet, nous allons introduire une classe Cellule qui permet de définir un objet contenant deux attributs :

- un attribut valeur définissant la valeur contenue dans cette "cellule" (nombre, texte, un autre objet de nature quelconque...) ;
- un attribut suivant définissant l'adresse d'un autre objet Cellule qui "suit" cet objet (qui est l'objet suivant dans l'ordre de la structure).




```
'''Implantation de la classe Cellule Elle sera utilisée pour les listes,
    les piles et les files.'''

class Cellule :


def __init__(self, valeur = None, suivant = None) :

    self.valeur = valeur
    self.suivant = suivant
```

Chaque valeur de la liste est encapsulé dans un objet Cellule (défini plus haut) qui contient une valeur ainsi que l'adresse de la cellule suivante

Une proposition très minimaliste d'implantation peut être :
On importe la classe "Cellule" que l'on a défini précédemment (et qui doit être dans le même dossier)

```
from Cellule import Cellule
class Liste :
def __init__(self) :

    ''' Crée une liste vide.

    L'attribut ''head'' est un objet Cellule qui définit la cellule
       en tête de la liste (premier élément de la liste)
    '''
    self.head = None


def estVide(self) :
    '''        Renvoie ``True`` si la liste est vide et ``False`` sinon.
    '''

    return self.head == None

def insererEnTete(self, element) :
    '''
    Paramètres          ----------
    element : N importe quel type
    Description : L élément à ajouter en tête de la liste          -------
          Ajoute un élément en tête de liste.
    '''

    nouvelle_cellule = Cellule(element, self.head)
    self.head = nouvelle_cellule

def tete(self) :
```

```
        '''        Renvoie la valeur de l élément en tête de liste.
        '''

        if not(self.estVide()) :
            return self.head.valeur


    def queue(self) :
        '''        Renvoie la liste privée de son premier élément (queue de
       la liste)
        '''
        subList = None
        if not(self.estVide()) :
            subList = Liste()
            subList.head = self.head.suivant
        return subList
```

Celle-ci peut être complétée par des méthodes implantant de nouvelles possibilités Par exemple :

- renvoyer la longueur de la liste ;
- accéder au élément d'une liste ;
- ajouter un élément à la fin de la liste ;
- rechercher un élément dans une liste en renvoyant "Vrai" si l'élément est présent, "Faux" sinon.

```
    def __len__(self) :
        '''        Renvoie le nombre d 'éléments de la liste.
        '''
        taille  = 0
        celluleCourante = self.head
        while(celluleCourante != None) :
            celluleCourante = celluleCourante.suivant
            taille += 1
        return taille

    def __getitem__(self, position) :

        '''        Paramètres          position : Entier positif
       Description : La position de l'élément qu'on veut obtenir dans la
       liste           Renvoie l'élément situé à la position spécifiée en
       paramètre dans la liste.

        '''
        assert position < len(self) ,'la position n existe pas dans la liste
        '


        celluleCourante = self.head
        for i in range(position) :
            celluleCourante = celluleCourante.suivant
        return celluleCourante.valeur

    def ajouterFin(self, element) :

        '''        Paramètres              element : type quelconque
          Description : L'élément à ajouter à la fin de la liste
       Ajoute un élément à la fin de la liste.
        '''
```

```
    nouvelle_cellule = Cellule(element, None)
    if self.estVide() :
        self.head = nouvelle_cellule
    else:
        celluleCourante = self.head
    while celluleCourante.suivant != None:

        celluleCourante=celluleCourante.suivant
        celluleCourante.suivant = nouvelle_cellule


def __contains__(self, element) :
    '''
    Paramètres ---------- element :
    type quelconque Description : L'élément qu'on souhaite vé
   rifier ------- Renvoie ``True`` si la liste contient l'
   élément et ``False`` sinon.
    '''
    trouve = False
    celluleCourante = self.head
    while celluleCourante != None :
        if celluleCourante.valeur == element :
            trouve = True
            celluleCourante = celluleCourante.suivant
    return trouve
```

6 Implémentation des piles utilisant la programmation objet

Chaque valeur contenue dans la pile est encapsulée dans un objet Cellule (défini plus haut) qui contient une valeur ainsi que l'adresse de la cellule suivante.Un attribut top est attaché à la pile. C'est en fait un objet de type *Cellule* correspondant au sommet.

À partir du sommet, on peut donc accéder à chaque valeur contenue dans la pile.L'empilement et le dépilement va donc consister à « mettre à jour » l'attribut top de cet objet.Une proposition d'implantation peut être :

```
from Cellule import Cellule

class Pile :
def __init__(self) :
    '''       Crée une pile vide.       L'attribut 'top' est un objet
   Cellule qui définit la cellule        constituant le'somme' de la
   pile
    .
    '''
    self.top = None

def estVide(self) :
'''       Renvoie ``True`` si la pile est vide et ``False`` sinon.
'''
    return self.top == None
def sommet(self) :
```

```
    ''' Renvoie la valeur de l'élément au sommet de la pile.
    '''
    if not(self.estVide()) :
        return self.top.valeur
    else :
        return None

def empiler(self, element) :
    '''        Paramètres                element : est de n'importe quel
   type            Description : L'élément à empiler sur la pile.
           Ajoute un élément au sommet de la pile.
    '''
    nouvelleCellule = Cellule(element, self.top)
    self.top = nouvelleCellule

def depiler(self) :
    '''
    Dépile et renvoie l'élément situé au sommet de la pile.
    '''
    if not(self.estVide()) :
        valeur = self.top.valeur
        self.top = self.top.suivant
        return valeur
    else :
        return None

def __len__(self) :
'''
Renvoie le nombre d'éléments de la pile.

    '''
    taille  = 0
    celluleCourante = self.top
    while(celluleCourante != None) :

    celluleCourante=celluleCourante.suivant
    taille += 1
    return taille
```

7 Implémentation des files utilisant la programmation objet

Chaque valeur contenue la file est encapsulé dans un objet Cellule (définit plus haut) qui contient une valeur ainsi que l'adresse de la cellule suivante.Deux attributs (de type Cellule) sont attachés à la file :
• un attribut head correspondant à la tête (sortie) de la file. C'est ici qu'on récupère les éléments qu'on retire de la file ;
• un attribut end correspondant au bout (entrée) de la file. C'est à partir de cet emplacement qu'on ajoute des éléments à la file.Ajouter un élément à la file consiste en une opération où l'on modifie l'attribut 'end' de la file en créant un nouvel objet Cellule et en affectant correctement les attributs de l'objet.
Retirer un élément consiste à modifier l'attribut head en récupérant les valeurs de la cellule correspondante et en réaffectant cet attribut à la cellule suivante. Le seul cas particulier auquel il faut faire attention est l'ajout du premier élément :

il correspond à la fois à l'attribut head et end (c'est à la fois la tête et le bout de la file, donc même cellule).Une proposition d'implantation peut être :

```
'''On importe la classe'Cellul' qu'on a défini précédemment.Doit être
   dans le même dossier
'''

from Cellule import Cellule
class File :
def __init__(self) :

    '''       Créé une file vide.        L'attribut'head' est un objet
   Cellule qui définit la cellule         c'nstituant la tête (sortie)
   de la file.        L'attribut 'en' est un objet Cellule qui définit
   la cellule         constituant le bout (entrée) de la file.
    '''
    self.head = None
    self.end = None

def estVide(self) :
    '''        Renvoie ``True`` si la file est vide et ``False`` sinon.
    '''
    return self.head == None


def tete(self) :

    '''
    Renvoie la valeur de l'élément en tête de la file (premier élément).
    '''
    if not(self.estVide()) :
      return self.head.valeur
    else :
      return None

def ajouter(self, element) :
    '''       Paramètres        _________        element : N'importe
   quel type            Description : L'élément à ajouter au bout de la
   file.        _______        Ajoute un élément au bout de la file.
    '''
    dernierCellule = Cellule(element, None)

    if(self.estVide()) :
    #Cas particulier si la file ne contient rien. La tête == Le bout de
   la file.

        self.head = dernierCellule
    else :
        self.end.suivant = dernierCellule
        self.end = dernierCellule

def retirerTete(self) :
    '''       Retire et renvoie l'élément situé à la tête de la file (
   premier élément).       '''
```

```
    if not(self.estVide()) :
        valeur = self.head.valeur
        self.head = self.head.suivant
    return valeur
    else :
        return None

def __len__(self) :
    ''''
      Renvoie le nombre d'éléments de la file.

    '''
    taille  = 0
    celluleCourante = self.head
    while(celluleCourante!=None):
        celluleCourante=celluleCourante.suivant
        taille += 1
    return taille
```