Chap 06. Piles et files

Livre p 127 – Chap 7 Piles et files

1. Une structure abstraite : les piles (LIFO)

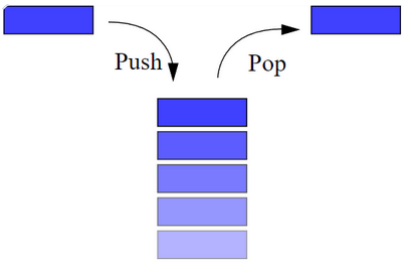
a. Notion de pile

Comme pour les listes chaînées, il s'agit d'étudier un modèle abstrait indépendant de tout langage de programmation.
Une **pile** (« stack » en anglais) est une structure de données linéaire fonctionnant sur le principe de : « Dernier arrivé, premier sorti » (« Last In, First Out » en anglais).
C'est d'ailleurs inspiré de la vie courante avec les piles d'assiettes par exemple !

b. Opérations élémentaires sur les piles

Les opérations élémentaires sur les listes piles sont :

- La **création** d'une pile vide.
- La **vérification** qu'une pile est vide ou non.
- L'**ajout** d'un élément au-dessus d'une pile.
  (**empiler**, « push » en anglais)
- La **lecture** puis **suppression** d'un élément sur le dessus d'une pile.
  (**dépiler**, « pop » en anglais)



2. Une structure abstraite : les files (FIFO)

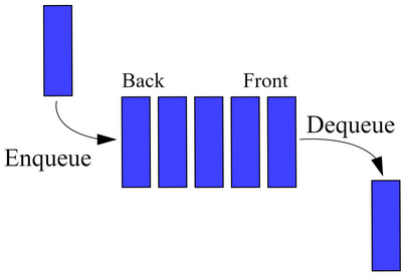
a. Notion de file

Il s'agit toujours d'étudier un modèle abstrait indépendant de tout langage de programmation.
Une **file** (« queue » en anglais) est une structure de données linéaire fonctionnant sur le principe de : « Premier arrivé, premier sorti » (« First In, First Out » en anglais).
C'est d'ailleurs inspiré de la vie courante avec les files d'attente par exemple !

b. Opérations élémentaires sur les files

Les opérations élémentaires sur les listes files sont :

- La **création** d'une file vide.
- La **vérification** qu'une file est vide ou non.
- L'**ajout** d'un élément à la fin d'une file.
  (**enfiler**, « enqueue » en anglais)
- La **lecture** puis **suppression** d'un élément au début d'une file.
  (**défiler**, « dequeue » en anglais)



3. Implémentation en Python

a. Avec le type list

En python, on peut utiliser le type list en tant que pile ou file, on retrouve les 4 opérations élémentaires.

Les **piles** :

- p = Pile() : p = []
- p.est_vide() : p == []
- p.empiler(a) : p.append(a)
- p.depiler() : p.pop()

Les **files** :

- f = File() : f = []
- f.est_vide() : f == []
- f.enfiler(a) : f= [a] + f
- f.defiler() : f.pop()

b. Avec des listes chaînées

L'utilisation du type list en python cache la vraie structure des piles et des files, on va donc repartir d'une autre structure abstraite : les listes chaînées :

- **Création** d'une **pile** vide.
  Exemple : pile_1 = creer_pile()
- **Ajout** d'un élément au-dessus d'une **pile**.
  Exemple : pile_1.empiler(5)
- **Lecture** puis **suppression** d'un élément sur le dessus d'une **pile**.
  Exemple : pile_1.depiler()

```
class Cellule:
    """une cellule d'une liste chaînée"""
    def __init__(self, v, s):
        self.valeur = v
        self.suivante = s

class Pile:
    """structure de pile"""
    def __init__(self):
        self.contenu = None

    def est_vide(self):
        return self.contenu is None

    def empiler(self, v):
        self.contenu = Cellule(v, self.contenu)

    def depiler(self):
        if self.est_vide():
            raise IndexError("pile vide")
        v = self.contenu.valeur
        self.contenu = self.contenu.suivante
        return v

    def __repr__(self):
        r = ""
        cel = self.contenu
        while not(cel is None):
            r += "{}\n".format(cel.valeur)
            cel = cel.suivante
        r += "---"
        return r

def creer_pile():
    return Pile()
```

- **Création** d'une **file** vide.
  Exemple : file_1 = creer_file()
- **Ajout** d'un élément à la fin d'une **file**.
  Exemple : file_1.enfiler(5)
- **Lecture** puis **suppression** d'un élément au début d'une **file**.
  Exemple : file_1.defiler()

```
class Cellule:
    """une cellule d'une liste chaînée"""
    def __init__(self, v, s):
        self.valeur = v
        self.suivante = s

class File:
    """structure de file"""
    def __init__(self):
        self.tete = None
        self.queue = None

    def est_vide(self):
        return self.tete is None

    def enfiler(self, x):
        c = Cellule(x, None)
        if self.est_vide():
            self.tete = c
        else:
            self.queue.suivante = c
        self.queue = c

    def defiler(self):
        if self.est_vide():
            raise IndexError("file vide")
        v = self.tete.valeur
        self.tete = self.tete.suivante
        if self.tete is None:
            self.queue = None
        return v

    def __repr__(self):
        r = ""
        cel = self.tete
        while not(cel is None):
            r = "{} -> ".format(cel.valeur) + r
            cel = cel.suivante
        return "-> " + r

def creer_file():
    return File()
```