Cours d'algorithmique et programmation 2

Anne Parrain

UFR des Sciences
Licence Sciences et Technologie
mentions Mathématiques et Informatique
Semestre 2

Section 1

Les files

La structure de données file (1)

La notion de *file* est peut-être plus naturelle qu'une pile : elle représente une file d'attente (à un guichet, par exemple).

Dans une file d'attente, les éléments arrivent dans un ordre et sortent dans le même ordre.

Une *file* est aussi appelée une structure **FIFO** : *First In First Out*.

La structure de données File (2)

De quoi a-t-on besoin ?

- d'une structure qui mémorise une collection de données
- d'une structure qui mémorise l'ordre d'ajout des données
- d'une structure qui garantisse l'enlèvement des éléments dans l'ordre de l'ajout

Les différences avec une pile sont liées à l'ajout et l'enlèvement des données. Les opérations élémentaires qu'on peut faire sur cette structure sont les mêmes que celles d'une pile, avec la même signature, mais pas la même sémantique.

Implémentation d'une classe `File`

```
1  class File :
2    '''Classe qui modelise la structure de donnees File.'''
3    def __init__(self):
4      '''File -> File
5      construit une file vide.'''
6    def ajouter(self,elt):
7      '''(modif) File, Objet -> None
8      ajoute elt a la fin de la file.'''
9    def enlever(self):
10     '''(modif) File -> Objet
11     enleve l'element en tete de la file
12     et le retourne.'''
13   def est_vide(self):
14     '''File -> boolean
15     teste si la pile est vide.'''
16   def sommet(self):
17     '''File -> Objet
18     retourne l'element en tete de la file
19     (sans l'enlever).'''
```

À vous !

Complétez l'implémentation de la classe `File`.

- Quel(s) attribut(s) nécessaire(s) ?
- Comment les utiliser ?

Implémentation d'une classe `File` (2)

```
1  class File :
2    '''Modelise une structure de donnees FIFO
3    (First In First Out)
4    '''
5
6    def __init__(self) :
7      '''File -> File
8      construit une file vide.'''
9      # basee sur une liste
10     self.__les_elts = list()
11
12   def ajoute(self, elt) :
13     '''(modif) File, Objet -> Rien
14     ajoute un element a la fin de la file.
15     '''
16     self.__les_elts.append(elt)
```

Implémentation d'une classe `File` (3)

```
1   def est_vide(self) :
2     '''File -> Boolean
3     teste si la file est vide.
4     '''
5     return len(self.__les_elts) == 0
6
7   def sommet(self) :
8     '''File -> Objet
9     retourne le premier element de la file.
10    '''
11    assert not self.est_vide()
12    return self.__les_elts[0]
13
14# fin de la classe File
```

Implémentation d'une classe `File` (4)

```
1
2   def enleve(self) :
3     '''(modif) File -> Objet
4     enleve le premier element de la file.
5     '''
6     assert not self.est_vide()
7     elt = self.__les_elts[0]
8     del(self.__les_elts[0])
9     return elt
```

Section 2

Utiliser une file

La bataille

La *bataille* est un jeu de cartes qui se joue à deux. Sans stratégie aucune, ce jeu s'apparente plutôt à une sorte de réussite.

Principe : un jeu de 52 cartes est distribué entre deux joueurs. Chaque joueur fait un tas de ses cartes et les considère face cachée. Chacun retourne sa première carte :

- la carte la plus forte (dans un ordre *As*, *Roi*, *Dame*, *Valet*, *10*, . . .*2*) remporte le pli, et le joueur qui vient de gagner pose ces cartes en-dessous de son tas
- si les deux cartes ont la même valeur, alors il y a *bataille* :
  - les deux joueurs posent chacun une carte, face cachée, sur leur carte
  - puis une autre carte face visible
  - si les deux cartes sont de même valeur, on recommence le processus
  - on s'arrête quand l'un des deux cartes est plus grande que l'autre (le joueur gagnant met toutes ses cartes en dessous de son tas).

À vous !

- Quelles sont les structures de données à prévoir pour le jeu de la bataille ?
- Spécifiez puis implémentez une fonction pour l'algorithme général du jeu
- Spécifiez puis implémentez une fonction pour l'algorithme spécifique de la bataille.