Lycée René CASSIN

NSI T*erminale*

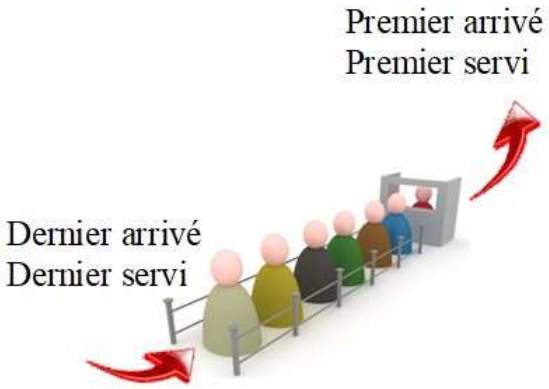
Structure de données - Les Files

Une structure de données est une organisation logique des données permettant de simplifier ou d'accélérer leur traitement.

Introduction

En informatique, une file (queue en anglais) est une structure de données basée sur le principe «Premier entré, premier sorti», en anglais FIFO (First In, First Out), ce qui veut dire que les premiers éléments ajoutés à la file seront les premiers à être récupérés.

Le fonctionnement ressemble à une file d'attente : les premières personnes à arriver sont les premières personnes à sortir de la file.



Voici quelques exemples d'usage courant d'une file :

- En général, on utilise des files pour mémoriser temporairement des transactions qui doivent attendre pour être traitées
- Les serveurs d'impression, qui doivent traiter les requêtes dans l'ordre dans lequel elles arrivent, et les insèrent dans une file d'attente (ou une queue).
- Certains moteurs multitâches, dans un système d'exploitation, qui doivent accorder du temps-machine à chaque tâche, sans en privilégier aucune

Pour implémenter une structure de file, on a besoin d'un nombre réduit d'opérations de bases :

- «enfiler»: ajoute un élément dans la file. Terme anglais correspondant : « enqueue »
- «défiler»: renvoie le prochain élément de la file, et le retire de la file. Terme anglais correspondant : « dequeue »
- «vide»: renvoie vrai si la file est vide, faux sinon
- «remplissage»: renvoie le nombre d'éléments dans la file.

La structure de file est un concept abstrait. Comment la réaliser dans la pratique ? Voici plusieurs implémentations possibles. L'idée principale étant que les fonctions de bases pourront être utilisées indépendamment de l'implémentation choisie.

Implémentation - Méthode 1

Nous utiliserons une simple liste pour représenter la file.

Là encore nous utiliserons append(x) et pop(0) pour réaliser les méthodes "enfiler" et "défiler"
Voici les fonctions de base:

```
def file():
    #retourne une file vide return []

def vide(f):
    ''' renvoie True si la file est vide
      False sinon''' return f==[]

def enfiler(f,x):
    #ajoute x à la file f" return
    f.append(x)

def defiler(f):
    #enlève et renvoie le premier élément de la file
    assert not vide(f), "file vide" return f.pop(0)
```

À faire1:

Tester les instructions suivantes :

```
f=file()
for i in range(5):
    enfiler(f,2*i)
print(f)
a=defiler(f)
print(a)
print(f)
print(vide(f))
```

Exercice1 :

Réaliser les fonctions taille(f) et sommet(f) qui retournent respectivement la taille de la file et le sommet de la file (le premier à sortir…) sans le supprimer

```
def taille(p):                          |   def sommet(p):
                                        |
                                        |
                                        |
                                        |
                                        |
-                                       |
```

Implémentation - Méthode 2

Nous allons créer une classe File pour implémenter cette structure.

À FAIRE 2:

En vous inspirant de ce que l'on a vu pour la classe `Pile()`, réaliser cette implémentation

```
class File:
    ''' classe File
    création d'une instance File avec une liste
    '''
    def __init__(self):
        "Initialisation d'une file vide"
        ...........

    def vide(self):
        "teste si la file est vide"
        ...........

    def defiler(self):
        "défile"
        ..........................
        ..........................

    def enfiler(self,x):
        "enfile"
        ..................
```

Écrire les instructions permettant de :

- créer une file
- la remplir avec les entiers 0,2,4,6,8
- la faire afficher
- "défiler" la file en faisant afficher l'élément récupéré

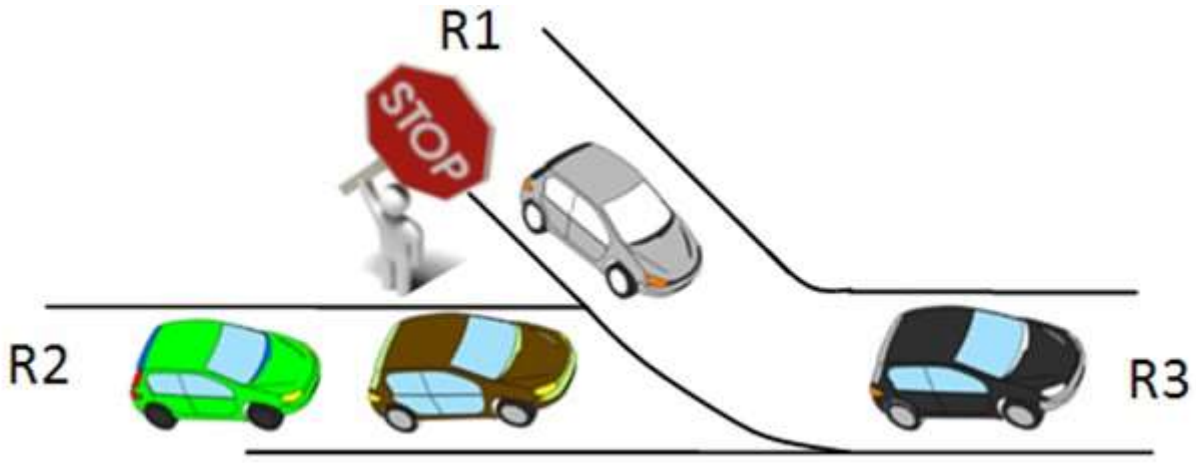
EXERCICE 2 :

Réaliser les méthodes `taille(self)` et `sommet(self)` qui retourne respectivement la taille de la file et le sommet de la file(le premier à sortir..) sans le supprimer

```
def taille(self):                |   def sommet(self):
                                 |
                                 |
                                 |
                                 |
                                 |
-                                |
```

Exemple - Croisement routier

Pour simuler un croisement routier, à sens unique, on utilise 3 files *f* 1, *f* 2 et *f* 3 représentant respectivement les voitures arrivant sur des routes R1 et R2, et les voitures partant sur la route R3. La route R2 a un STOP. Les voitures de la file *f* 2 ne peuvent avancer que s'il n'y a aucune voiture sur la route R1, donc dans la file *f* 1.



On souhaite écrire un algorithme qui simule le départ des voitures sur la route R3, modélisée par la file *f* 3.

- Dans la file *f* 1 on représentera la présence d'une voiture par le nombre 1 et l'absence de voiture par 0
- Dans la file *f* 2 on représentera la présence d'une voiture par le nombre 2 et l'absence de voiture par 0
- On n'utilisera que les méthodes enfiler, defiler, sommet et vide
- On testera l'algorithme sur *f* 1: tête <–[0, 1, 1, 0, 1]<– queue
- On testera l'algorithme sur *f* 2: tête <–[0, 2, 2, 2, 0, 2, 0]<– queue
- Le résultat attendu: *f* 3 tête <–[0, 1, 1, 2, 1, 2, 2, 0, 2, 0]<– queue

Question1:

Que doit faire l'algorithme si les deux sommets des files sont à 0?

.....
.....
.....
.....

Question2:

Que doit faire l'algorithme si le sommet de *f* 1 est à 1 et celui de *f* 2 à 2?

.....
.....
.....
.....

Question3:

Que doit faire l'algorithme si le sommet de *f* 1 est à 1 et celui de *f* 2 à 0?

.....
.....
.....
.....

Question4:

Que doit faire l'algorithme si le sommet de *f* 1 est à 0 et celui de *f* 2 à 2?

.....
.....
.....
.....

Question5:

Que doit faire l'algorithme si l'une des deux files est vide?

.....
.....
.....
.....

Question6:

Écrire un algorithme qui modélise ce carrefour, on utilisera une fonction croisement(f1,f2) qui prend en paramètres deux files *f* 1 et *f* 2 et qui retourne une file *f* 3 contenant la file *f* 3 des voitures sur la route R3

.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....

.....
.....
.....

Exercice3 :

Réaliser le programme et tester le dans différents scénarios

Prolongement possible

Complexifier la situation en imaginant des croisements successifs de trois ou quatre routes avec des STOP et des priorités à droite...

Exercices

Exercice4 :

On dispose d'une file, écrire une fonction qui renvoie la file inversée (l'élément de la tête sera situé à la queue et ainsi de suite…)

On utilisera une pile et seulement les méthodes associées aux piles et aux files

Exercice5 :

On dispose d'une file contenant des entiers, écrire une fonction qui renvoie une file où on aura séparé les nombres pairs des impairs

Exercice6 :

On dispose d'une pile contenant des entiers, écrire une fonction ou une méthode qui renvoie une pile où l'on aura supprimé le premier ou le dernier élément entré