DS1

Exercice : Cours

1. Écrire une fonction **Python** calculant la première position du maximum d'une liste `L`.

2. *Tri fusion*

a) Implémenter la fonction `fusion` qui prend en paramètre deux listes triées `L1` et `L2` et renvoie la liste `L` qui réalise la fusion des deux listes précédentes.

b) Implémenter la fonction `tri_fusion` qui prend en paramètre une liste `L`, trie cette liste selon le principe du tri fusion et renvoie la liste obtenue.

Problème : Trafic routier

Ce problème concerne la conception d'un logiciel d'étude de trafic routier. On modélise le déplacement d'un ensemble de voitures sur des files à sens unique (voir Figure 1 et 2). C'est un modèle simple qui peut permettre de comprendre l'apparition d'embouteillages et de concevoir des solutions pour fluidifier le trafic.

Notations :
Soit `L` une liste. Soit `p` $\in \mathbb{N}$.

- on note `len(L)` sa longueur ;
- pour tout `i` $\in [\![0, \texttt{len(L)} - 1]\!]$, l'élément de la liste d'indice i est noté `L[i]` ;
- pour tout `(i, j)` $\in [\![0, \texttt{len(L)}]\!]$ tel que : $i < j$, on note `L[i : j]` la sous-liste composée des éléments `L[i]`, ..., `L[j - 1]` ;
- `L * p` est la liste obtenue en concaténant `p` copies de `L`. Par exemple, `[0] * 3` est la liste `[0, 0, 0]`.

Figure 1 Représentation d'une file de longueur onze comprenant quatre voitures, situées respectivement sur les cases d'indices 0, 2, 3 et 10.

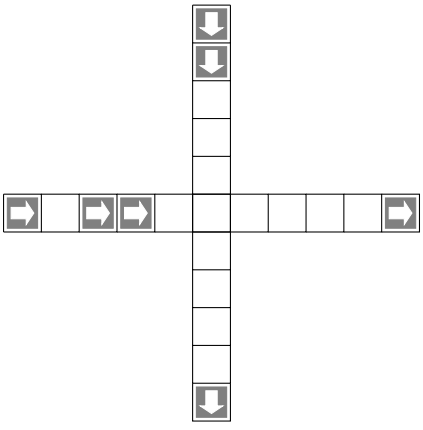


Figure 2 Configuration représentant deux files de circulation à sens unique se croisant en une case. Les voitures sont représentées par un carré gris.

Partie 1 : Préliminaires

Dans un premier temps, on considère le cas d'une seule file, illustré par la Figure 1. Une file de longueur n est représentée par n cases. Une case peut contenir au plus une voiture. Les voitures présentes dans une file circulent toutes dans la même direction (sens des indices croissants, désigné par les flèches sur la Figure 1) et sont indifférenciées.

1. Expliquer comment représenter une file de voitures à l'aide d'une liste de booléens.

2. Donner une instruction **Python** permettant de définir une liste `A` représentant la file de voitures illustrée par la Figure 1.

3. Soit `L` une liste représentant une file de longueur n. Soit `i` $\in [\![0, n-1]\!]$. Définir en **Python** la fonction `occupe` de paramètres `L` et `i` qui renvoie `True` lorsque la case d'indice `i` de la file est occupée par une voiture et `False` sinon.

4. Combien existe-t-il de files différentes de longueur n ? Justifier votre réponse.

5. Écrire une fonction `egal` de paramètres `L1` et `L2` retournant un booléen permettant de savoir si deux listes `L1` et `L2` sont égales.

6. Quelle est la complexité, dans le pire cas, de la fonction `egal` ? Autrement dit, combien de comparaisons de booléens cette fonction effectue-t-elle, au maximum ?

7. Préciser le type d'objet renvoyé la fonction `egal`.

Partie 2 : Déplacement de voitures dans la file

On identifie désormais une file de voitures à une liste. On considère les schémas de la Figure 3 représentant des exemples de files. Une étape de simulation pour une file consiste à déplacer les voitures de la file, à tour de rôle, en commençant par la voiture la plus à droite, d'après les règles suivantes :

- une voiture se trouvant sur la case la plus à droite de la file sort de la file ;
- une voiture peut avancer d'une case vers la droite si elle arrive sur une case inoccupée ;
- une case libérée par une voiture devient inoccupée ;
- la case la plus à gauche peut devenir occupée ou non, selon le cas considéré.

On suppose avoir écrit en **Python** la fonction `avancer` prenant en paramètres une liste de départ `L`, un booléen `b` indiquant si la case la plus à gauche doit devenir occupée lors de l'étape de simulation, et renvoyant la liste obtenue par une étape de simulation.

Par exemple, l'application de cette fonction à la liste illustrée par la Figure 3(a) permet d'obtenir soit la liste illustrée par la Figure 3(b) lorsque l'on considère qu'aucune voiture nouvelle n'est introduite, soit la liste illustrée par la Figure 3(c) lorsque l'on considère qu'une voiture nouvelle est introduite.

(a) Liste initiale `A`

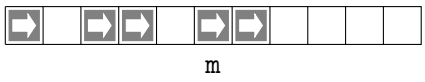
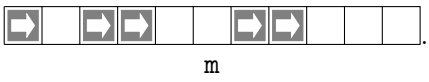
(b) `B = avancer(A, False)`

(c) `C = avancer(A, True)`

Figure 3 Étape de simulation

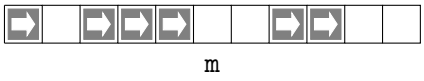
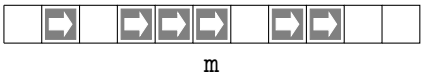
8. Pour la liste `A` définie à la question ***2.***, que renvoie `avancer(avancer(A, False), True)` ?

9. On considère `L` une liste et `m` l'indice d'une case de cette liste ($\texttt{m} \in [\![0, \texttt{len(L)} - 1]\!]$). On s'intéresse à une étape partielle où seules les voitures situées sur la case d'indice `m` ou à droite de cette case peuvent avancer normalement, les autres voitures ne se déplaçant pas.

Par exemple, la file  devient  .

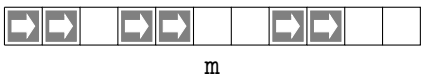
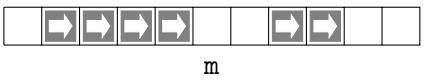
Définir en **Python** la fonction `avancer_fin` de paramètres `L` et `m` qui réalise cette étape partielle de déplacement et renvoie le résultat dans une nouvelle liste sans modifier `L`.

10. Soient `L` une liste, `b` un booléen et `m` l'indice d'une case inoccupée de cette liste. On considère une étape partielle où seules les voitures situées à gauche de la case d'indice `m` se déplacent, les autres voitures ne se déplacent pas. Le booléen `b` indique si une nouvelle voiture est introduite sur la case la plus à gauche.

Par exemple, la file  devient  lorsque aucune nouvelle voiture n'est introduite.

Définir en **Python** la fonction `avancer_debut` de paramètres `L`, `b` et `m` qui réalise cette étape partielle de déplacement et renvoie le résultat dans une nouvelle liste sans modifier `L`.

11. On considère une liste `L` dont la case d'indice `m` > 0 est temporairement inaccessible et bloque l'avancée des voitures. Une voiture située immédiatement à gauche de la case d'indice `m` ne peut pas avancer. Les voitures situées sur les cases plus à gauche peuvent avancer, à moins d'être bloquées par une case occupée, les autres voitures ne se déplacent pas. Un booléen `b` indique si une nouvelle voiture est introduite lorsque cela est possible.

Par exemple, la file  devient  lorsque aucune nouvelle voiture n'est introduite.

Définir en **Python** la fonction `avancer_debut_bloque` de paramètres `L`, `b` et `m` qui réalise cette étape partielle de déplacement et renvoie le résultat dans une nouvelle liste.

Partie 3 : Une étape de simulation à deux files

On considère dorénavant deux files `L1` et `L2` de même longueur impaire se croisant en leur milieu. On note `m` l'indice de la case du milieu.

- La file `L1` est toujours prioritaire sur la file `L2`.
- Les voitures ne peuvent pas quitter leur file et la case de croisement ne peut être occupée que par une seule voiture.
- Les voitures de la file `L2` ne peuvent accéder au croisement que si une voiture de la file `L1` ne s'apprête pas à y accéder.
- Une étape de simulation à deux files se déroule en deux temps.
  - × Dans un premier temps, on déplace toutes les voitures situées sur le croisement ou après.
  - × Dans un second temps, les voitures situées avant le croisement sont déplacées en respectant la priorité.

Par exemple, partant d'une configuration donnée par la Figure 4, les configurations successives sont données par les Figures 4(b), 4(c), 4(d), 4(e) et 4(f) en considérant qu'aucune nouvelle voiture n'est introduite.

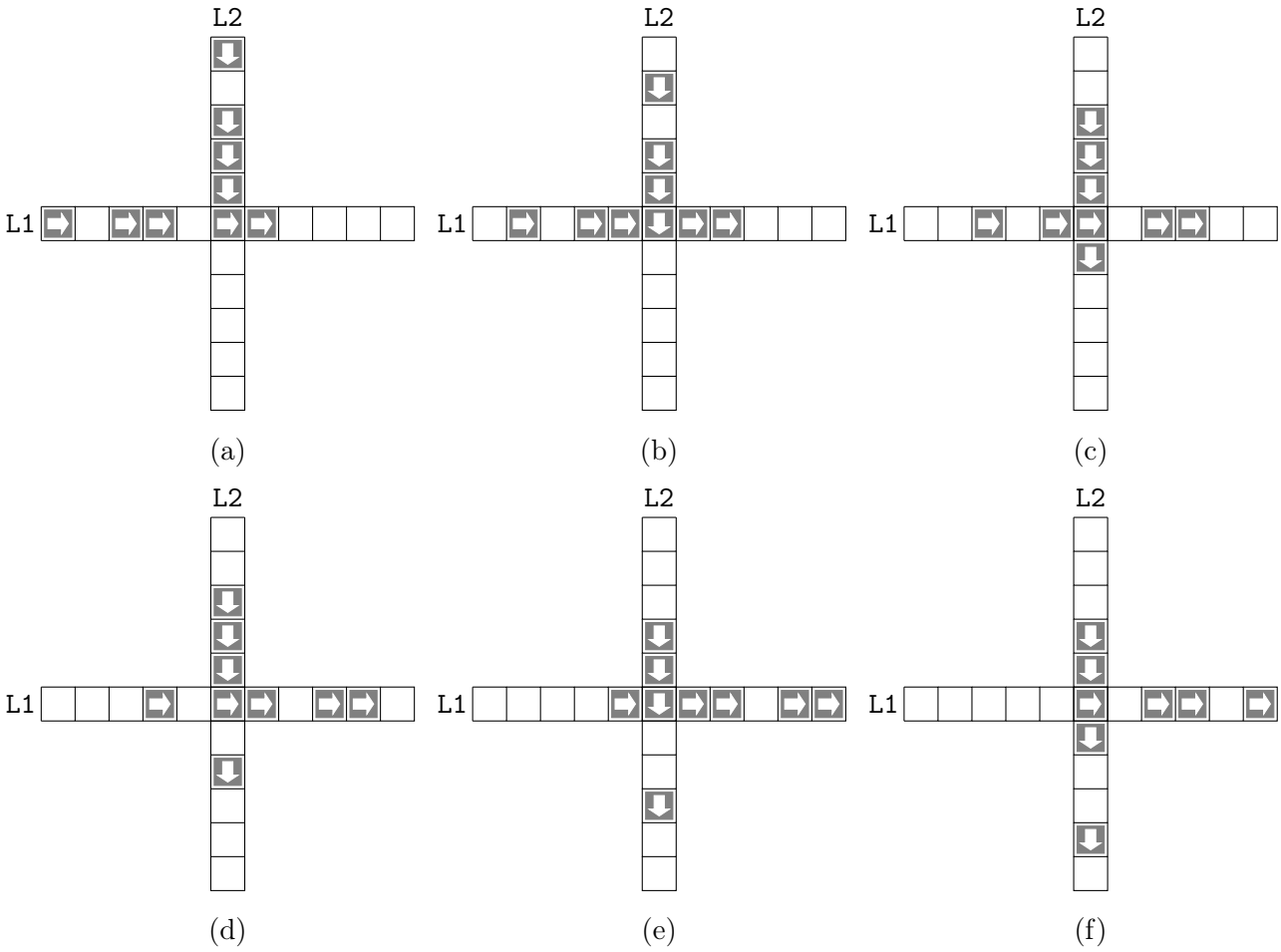


Figure 4 Étapes de simulation à deux files

L'objectif de cette partie est de définir en **Python** l'algorithme permettant d'effectuer une étape de simulation pour ce système à deux files.

12. En utilisant le langage **Python**, définir la fonction `avancer_files` de paramètres `L1`, `b1`, `L2` et `b2` qui renvoie le résultat d'une étape de simulation sous la forme d'une liste de deux éléments notée `[R1,R2]` sans changer les listes `L1` et `L2`. Les booléens `b1` et `b2` indiquent respectivement si une nouvelle voiture est introduite dans les files `L1` et `L2`. Les listes `R1` et `R2` correspondent aux listes après déplacement.

13. On considère les listes

```
D = [False,True,False,True,False]
E = [False,True,True,False,False]
```

Que renvoie l'appel `avancer_files(D, False, E, False)` ?