Corrigé Devoir surveillé n°1
Informatique Tronc Commun le 14/12/24

Etude de trafic routier

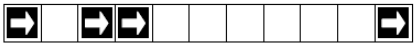
Ce sujet concerne la conception d'un logiciel d'étude de trafic routier. On modélise le déplacement d'un ensemble de voitures sur des files à sens unique (voir Figure 1(a) et 1(b)). C'est un schéma simple qui peut permettre de comprendre l'apparition d'embouteillages et de concevoir des solutions pour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

Le sujet comporte des questions de programmation. Le langage à utiliser est Python.

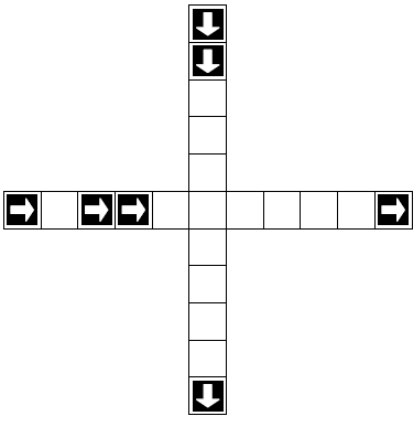
Notations :

Soit L une liste,

- on note len(L) sa longueur ;
- pour i entier, $0 \leq i < \text{len}(L)$, l'élément de la liste d'indice i est noté L[i] ;
- pour i et j entiers, $0 \leq i < j \leq \text{len}(L)$, L[i : j] est la sous-liste composée des éléments L[i], . . ., L[j − 1] ;
- p * L, avec p entier, est la liste obtenue en concaténant p copies de L. Par exemple, 3* [0] est la liste [0, 0, 0].



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



(b) Configuration représentant deux files de circulation à sens unique se croisant en une case. Les voitures sont représentées par un carré noir.

FIGURE 1 – Files de circulation

Partie I Préliminaires

/ ≃ 15

Dans un premier temps, on considère le cas d'une seule file, illustré par la Figure 1(a). 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(a)) et sont indifférenciées.

1) Les éléments de la liste sont True ou False, True si la case contiznt une voiture, False si elle est vide.

2) La liste A doit être : [True, False, True, True, False, False, False, False, False, False, True].

On peut écrire : A = [True, False, True, True, False, False, False, False, False, False, True]

Ou A = [[True, False, True, True] + 6* [False] + [True]

3) Soit L une liste représentant une file de longueur n et i un entier tel que $0 \leq i < n$. Définissons en Python la fonction occupe(L, i) qui renvoie True lorsque la case d'indice i de la file est occupée par une voiture et False sinon :

```
def occupe(L, i) :
'" renvoie True si la case d'indice i de la file représentée par L contient une voiture
, False sinon"'
    return L[i]
```

4) Pour chaque case il y a 2 possibilités, il y a n cases donc il y a $\mathbf{2^n}$ possibilités.

5) Ecrivons une fonction egal(L1, L2) retournant un booléen permettant de savoir si deux listes L1 et L2 sont égales.

```
def egal (L1, L2) :
'"teste si les deux listes sont identiques"'
    return L1 == L2
```

ou

```
def egal (L1, L2) :
'"teste si les deux listes sont identiques"'
    if len(L1) != len(L2) :
        return False
    for i in range(len(L1)) :
        if L1[i] != L2[i] :
            return False
    return True
```

6) Le type de retour de cette fonction est un booléen : bool en Python.

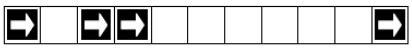
Partie II Déplacement de voitures dans la file / ≃ 5

On identifie désormais une file de voitures à une liste. On considère les schémas de la Figure 2 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.

7) On se propose d'écrire en Python la fonction avancer prenant en paramètres une liste de départ, un booléen 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 2(a) permet d'obtenir soit la liste illustrée par la Figure 2(b) lorsque l'on considère qu'aucune voiture nouvelle n'est introduite, soit la liste illustrée par la Figure 2(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 2 – Étape de simulation

Définissons en Python la fonction avancer (L, b) qui réalise cette étape partielle de déplacement et renvoie la liste modifiée sans modifier L.

```
def avancer (L, b):
'''avance les cases de L, la 1ere sera b'''
  return [b] + L[:len(L)-1]
```

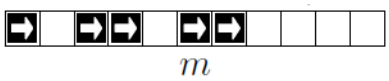
Etant donnée A la liste définie à la question 2, avancer(avancer(A, False),True) renvoie :

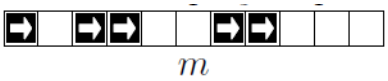
[True, False, True, False, True, True, False, False, False, False, False]

Correspond à 

car avancer(A,False) renvoie : [False, True, False, True, True, False, False, False, False, False, False]]

8) On considère L une liste et m l'indice d'une case de cette liste ($0 \leq m < len(L)$). 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 

Définissons en Python la fonction avancer_fin(L, m) qui réalise cette étape partielle de déplacement et renvoie le résultat dans une nouvelle liste sans modifier L :

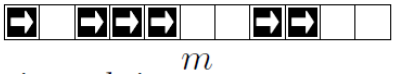
```
def avancer_fin (L, m) :
'''avance les cases m et à droite'''
  L1 = L[ : ]
  for i in range (m, len(L1)-1) :
      L1[i+1] = L[i] :
  L1[m] = False
  return L1
```

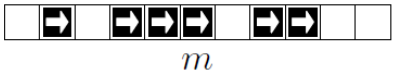
Ou en utilisant avancer :

```
def avancer_fin (L, m) :
'''avance les cases m et à droite'''
  return L[ :m] + avancer(L[m : ], False)
```

9) 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.

Définissons en Python la fonction avancer_debut(L, b, m) qui réalise cette étape partielle de déplacement et renvoie le résultat dans une nouvelle liste sans modifier L.

Version itérative :

```
def avancer_debut (L, b, m) :
'''avance les cases à gauche de m et 1ere case = b'''
  L1 = L[ : ]
  for i in range (1, m+1) :
      L1[i] = L[i-1] :
  L1[0] = b
  return L1
```

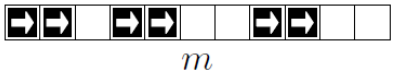
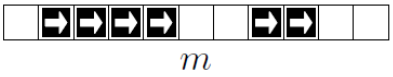
Ou :

```
def avancer_debut (L, b, m) :
'''avance les cases à gauche de m et 1ere case = b'''
  return [b] + L[ :m] + L[m+1 :]
```

Ou en utilisant avancer :

```
def avancer_debut (L, b, m) :
'''avance les cases à gauche de m et 1ere case = b'''
  return avancer( L[ :m+1], b) + L[m+1 :]
```

10) 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.

Définissons en Python la fonction avancer_debut_bloque(L, b, m) qui réalise cette étape partielle de déplacement et renvoie le résultat dans une nouvelle liste.

En utilisant avancer_debut

```
def avancer_debut_bloque (L, b, m) :
'''avance les cases à gauche de m si elles ne sont pas bloquées'''
  for i in range (m-1,-1,-1) :  # i va de m-1 à 0 de -1 en -1
    if L[i]==False:
      return avancer_debut(L,b,i)
  return L[:]
```

Version itérative :

```
def avancer_debut_bloque (L, b, m):
  L2 = L[:]  # Copier L
  for i in range (m-1, 0, -1) :   # i va de m-1 à 1 de -1 en -1
      if L2[i] == False:  # pas de voiture case i
          L2[i] = L2[i-1]  # ça avance
          L2[i-1] = False
  L2[0] = b or L2[0]
  return L2
```

Partie III Une étape de simulation à deux files / ≃5

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 3(a), les configurations successives sont données par les Figures 3(b), 3(c), 3(d), 3(e) et 3(f) en considérant qu'aucune nouvelle voiture n'est introduite.

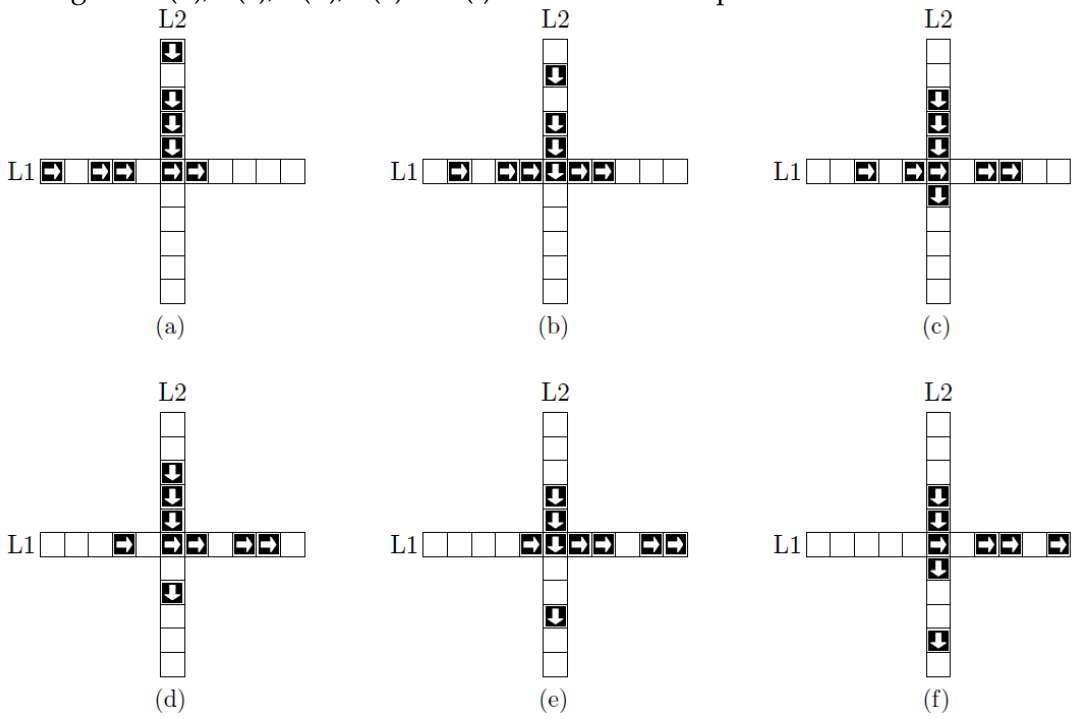


FIGURE 3 – Étapes de simulation à deux files

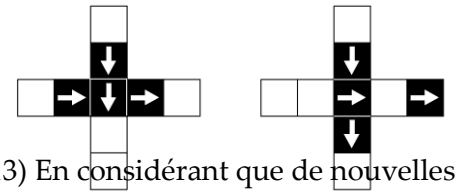
L'objectif de cette partie est de définir en Python l'algorithme permettant d'effectuer une étape de simulation pour ce système à deux files.

11) En utilisant le langage Python, définir la fonction 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.

```
def avancer_files(L1, b1, L2, b2) :
    m = len(L1)//2
    R2 = avancer_fin(L2, m)
    R1 = avancer(L1, b1)        # Les voitures de L1 ne peuvent pas être bloquées
    if occupe(R1,m):
        R2 = avancer_debut_bloque(R2, b2, m)
    else:
        R2 = avancer_debut(R2, b2, m)
    return [R1, R2]
```

12) 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) ?

Les files évoluent comme suit :



L'appel renvoie :

[[False, False, True, False, True], [False, True, False, True, False]]

13) En considérant que de nouvelles voitures peuvent être introduites sur les premières cases des files lors d'une étape de simulation, décrire une situation où une voiture de la file L2 serait indéfiniment bloquée :

Considérons la situation suivante :

- La file L1 est pleine,
- à chaque étape de la simulation, on ajoute une nouvelle voiture à la file L1,
- une voiture est sur la case m − 1 de la file L2.

La voiture sur la case m − 1 de la file L2 est indéfiniment bloquée.

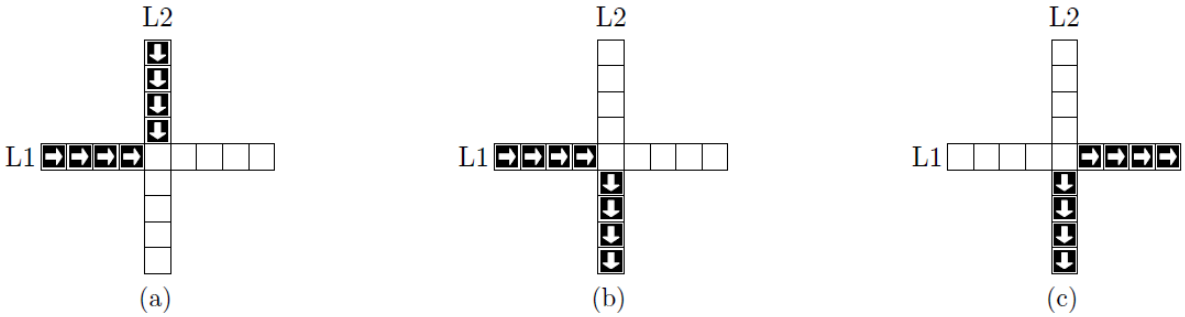


FIGURE 4 – Étude de configurations

14) Etant données les configurations illustrées par la Figure 4, combien d'étapes sont nécessaires (on demande le nombre minimum) pour passer de la configuration 4(a) à la configuration 4(b) ?

- Les voitures de la file L2 doivent laisser la priorité aux voitures de la file L1, elles restent donc immobiles jusqu'à l'étape 4 inclue.
- Les voitures de la file L2 commencent à se déplacer à l'étape 5 et arrive dans la position voulue à l'étape 9.

Il n'est donc pas possible d'atteindre la configuration demandée en moins de 9 étapes.

Il suffit d'ajouter des voitures à L1 aux étapes 6, 7, 8 et 9 (et de n'ajouter aucune voiture à L2) pour obtenir la configuration souhaitée en 9 étapes.

Conclusion : il faut **9 étapes minimum**.

15) Peut-on passer de la configuration 4(a) à la configuration 4(c) ? Justifier votre réponse.
On ne peut pas arriver à la configuration 4(c) à partir de 4(a) (ni à partir d'aucune autre d'ailleurs), car à l'étape précédente, il faudrait nécessairement que les 4 voitures des deux files soit décalées d'une case à gauche et en haut pour qu'il en reste 4 ensuite, et donc deux voitures occuperaient le croisement ce qui n'est pas possible.

Conclusion : il est **impossible** de passer de la configuration 4(a) à la configuration 4(c).