ISM MPSI/PCSI, Informatique
Année 2023/2024

Concours blanc - Informatique
Lundi 3 juin 2024 - 13h30-15h30

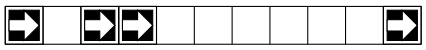
L'utilisation des calculatrices n'est pas autorisée pendant cette épreuve.
On attachera une grande importance à la concision, à la clarté et à la précision de la rédaction. L'indentation devra être matérialisée par un trait vertical et les codes non triviaux devront impérativement être commentés.

Étude de trafic routier

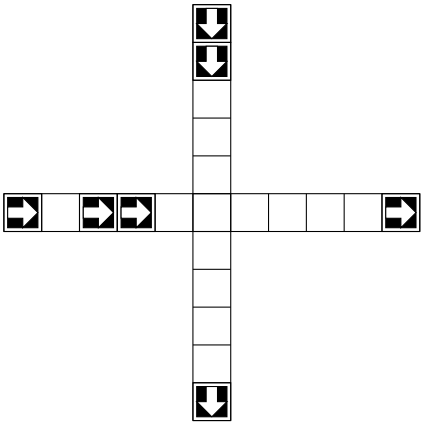
Ce sujet concerne la conception d'un logiciel d'étude de trafic routier. On modélise le déplacement d'un ensemble de voitures sur des files à sens unique (voir figures 1.(a) et 1.(b)). C'est un schéma simple qui peut permettre de comprendre l'apparition d'embouteillages et de concevoir des solutions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

Rappels

En langage python, on rappelle les instructions suivantes :
- `L1+L2` réalise la concaténation de deux listes `L1` et `L2` ;
- `[False]*n` avec `n` entier renvoie la liste `[False, False, ...]` de taille `n` ;
- `L[i:j]` renvoie la liste `[L[i], ..., L[j-1]]`, `L[i:]` renvoie la liste `L[i:len(L)]` et `L[:j]` renvoie la liste `L[0:j]`.



(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

I Préliminaires

Dans un premier temps, on considère le cas d'une 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 . Expliquer comment représenter une file de voitures à l'aide d'une liste de booléens.

2 . Donner une ou plusieurs instructions python permettant de définir une liste `A` représentant la file de voitures illustrée par la figure 1.(a).

3 . Écrire une fonction `occupe(L,i)` d'arguments `L` une liste de longueur n représentant une file de voitures et `i` un entier dans $[\![0\,;\,n-1]\!]$ qui renvoie `True` si la case d'indice `i` de la file `L` est occupée par une voiture et `False` sinon.

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

II Déplacement de voitures dans la file

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.

5 . Écrire une fonction `avancer(L,b)` d'arguments `L` une liste représentant une file de voitures et `b` un booléen qui renvoie une nouvelle liste représentant une file de voitures après une étape de simulation, l'état de la case la plus à gauche étant déterminé par le booléen `b`. 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) si aucune nouvelle voiture n'est introduite, soit la liste illustrée par la figure 2.(c) si une nouvelle voiture est introduite.

(a) Liste initiale `A`

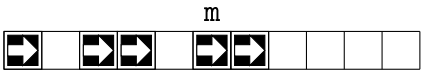
(b) `B=avancer(A,False)`

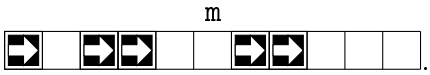
(c) `C=avancer(A,True)`

FIGURE 2 – Étape de simulation

6 . Étant donnée `A` la liste définie à la question 2, que renvoie `avancer(avancer(A,False),True)` ?

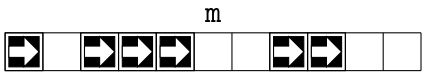
7 . On considère `L` une liste et `m` l'indice d'une case de cette liste ($m \in [\![0\,;\,len(\mathrm{L})-1]\!]$). On s'intéresse à une étape partielle où seules les voitures situées sur la case d'indice `m` ou à droite de cette case peuvent avancer normalement, les autres voitures ne se déplaçant pas.

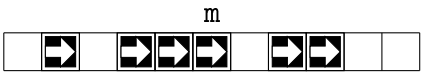
Par exemple, la file 

devient 

.
Écrire une fonction `avancer_fin(L,m)` qui réalise cette étape partielle de déplacement et renvoie le résultat dans une nouvelle liste sans modifier `L`.

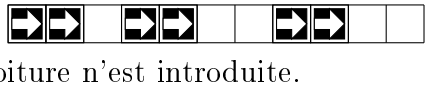
8 . Soit `L` une liste, `b` un booléen et `m` l'indice d'une case inoccupée de cette liste. On considère une étape partielle où seules les voitures situées à gauche de la case d'indice `m` se déplacent, les autres voitures ne se déplaçant pas. Le booléen `b` indique si une nouvelle voiture est introduite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

Par exemple, la file 

devient 

lorsque aucune nouvelle voiture n'est introduite.
Écrire une fonction `avancer_debut(L,b,m)` qui réalise cette étape partielle de déplacement et renvoie le résultat dans une nouvelle liste sans modifier `L`.

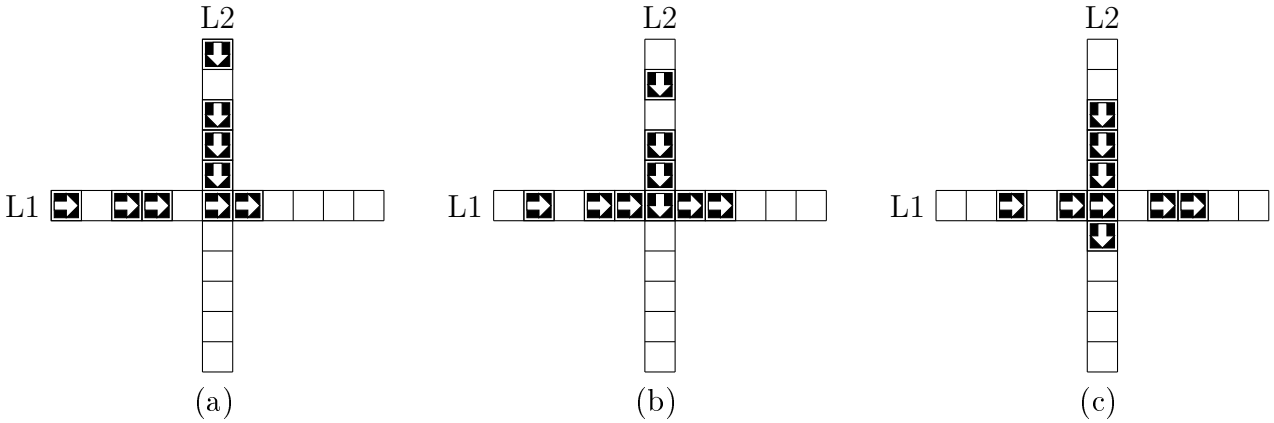
9 . On considère une liste `L` dont la case d'indice `m` avec $m > 0$ est temporairement inaccessible et bloque l'avancée des voitures. Une voiture située immédiatement à gauche de la case d'indice `m` ne peut pas avancer. Les voitures situées sur les cases plus à gauche peuvent avancer, à moins d'être 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.
Écrire une fonction `avancer_debut_bloquer(L,b,m)` qui réalise cette étape partielle de déplacement et renvoie le résultat dans une nouvelle liste. L'usage des fonctions `occupe` et `avance_debut` est impératif.

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.



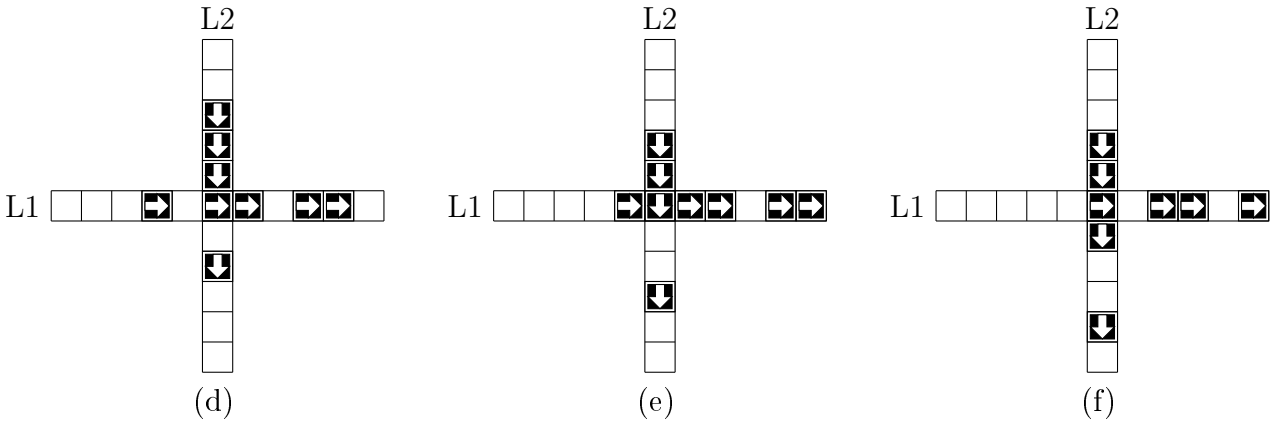


FIGURE 3 – Étape de simulation à deux files

III Une étape de simulation à deux files

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

10 . Écrire une fonction `avancer_files(L1,b1,L2,b2)` d'arguments `L1`, `L2` des listes, `b1`, `b2` des booléens qui renvoie le résultat d'une étape de simulation pour les files de voitures représentées par `L1` et `L2`. Le résultat renvoyé est une liste de deux listes `[R1, R2]` correspondant aux files après déplacement. Les listes `L1`, `L2` ne sont pas modifiées. Les booléens `b1` et `b2` indiquent respectivement si une nouvelle voiture est introduite dans les files `L1` et `L2`. L'usage des fonctions `occupe`, `avancer`, `avancer_debut`, `avancer_fin`, `avancer_debut_bloquer` est impératif.

11 . 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)` ? Compléter votre réponse par une figure illustrant les configurations avant puis après l'appel de la fonction.

IV Transitions

12 . 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.

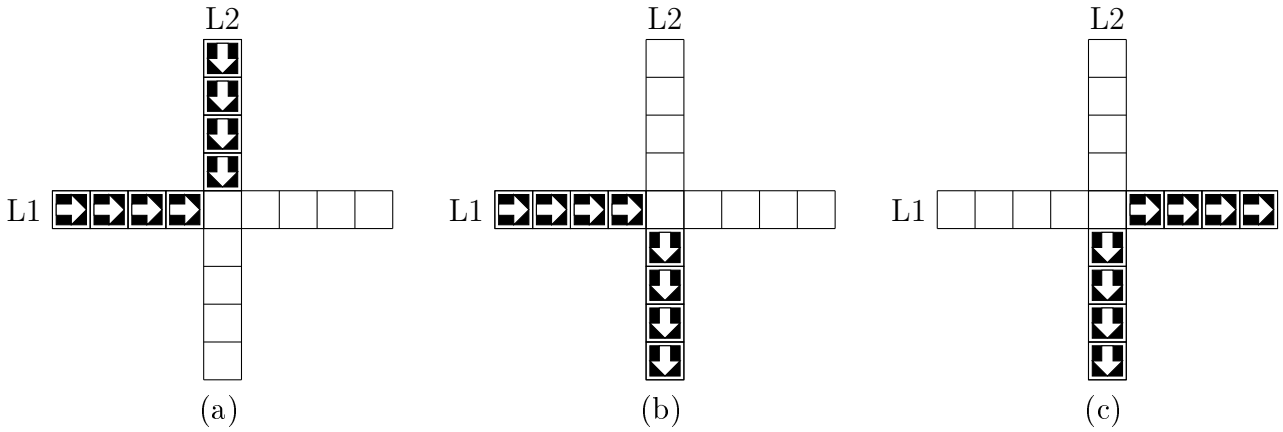


FIGURE 4 – Étude de configurations

13 . Étant 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) ? Justifier votre réponse.

14 . Peut-on passer de la configuration 4.(a) à la configuration 4.(c) ? Justifier votre réponse.

V Atteignabilité

Certaines configurations peuvent être néfastes pour la fluidité du trafic. Une fois ces configurations identifiées, il est intéressant de savoir si elles peuvent apparaître. Lorsque c'est le cas, on dit qu'une telle configuration est *atteignable*.

Pour savoir si une configuration est atteignable à partir d'une configuration initiale, on a écrit le code incomplet fourni en annexe.

15 . Écrire une fonction `tri(L)` d'argument `L` une liste d'objets comparables avec l'opérateur usuel `<` qui renvoie la liste triée. Le tri pourra ne pas être en place mais il devra toutefois être performant. L'usage de la fonction `sorted` ou de la méthode `sort` sont interdits.

Le langage python sait comparer deux listes de booléens à l'aide de l'opérateur usuel `<` et on peut donc utiliser la fonction `tri` précédemment écrite sur une liste de liste de booléens.

16 . Écrire une fonction non récursive `elim_double(L)` d'argument `L` une liste d'objets triée qui renvoie la liste triée obtenue par élimination des doublons et de complexité linéaire en la taille de `L`. Par exemple, l'appel `elim_double([1, 1, 3, 3, 3, 7])` doit renvoyer la liste `[1, 3, 7]`.

On dispose de la fonction suivante :

```
def doublons(L):
    if len(L)>1:
        if L[0]!=L[1]:
            return [L[0]] + doublons(L[1:])
        return doublons(L[1:])
    else:
        return L
```

17 . Que renvoie l'appel `doublons([1, 1, 2, 2, 3, 3, 3, 5])` ?

18 . Cette fonction est-elle utilisable pour éliminer les éléments apparaissant plusieurs fois dans une liste non triée ? Justifier.

19 . La fonction `recherche` fournie en annexe permet d'établir si la configuration correspondant à `but` est atteignable en partant de l'état `init`. Préciser le type de retour de la fonction `recherche`, le type des variables `but` et `espace` ainsi que le type de retour de la fonction `successeurs`.

20 . Montrer qu'un appel à la fonction `recherche` de l'annexe se termine toujours.

21 . Proposer, en déplaçant l'instruction `stop = (ancien==espace)`, une amélioration simple de la fonction `recherche` .

VI Amélioration

22 . Afin d'améliorer l'efficacité du test `if but in espace` en ligne 10 de l'annexe, on propose de le remplacer par `if in_triee(but,espace)` avec une fonction `in_triee(elt,L)` qui réalise efficacement la recherche de `elt` dans liste triée `L`. Écrire le code de cette fonction `in_triee`.

23 . Afin de comparer plus efficacement les files représentées par des listes de booléens, on remarque que ces listes représentent un codage binaire où `True` correspond à 1 et `False` à 0. Écrire la fonction `versEntier(L)` d'argument `L` une liste de booléens qui renvoie l'entier correspondant. Par exemple, l'appel `versEntier([True, False, False])` renverra 4.

24 . On veut écrire la fonction inverse de `versEntier` transformant un entier en liste de booléens dont la taille est fixée. Pour un entier n non nul, que doit être au minimum la taille de la liste pour que le codage soit fidèle ? Écrire cette fonction `versFile(n,taille)` d'arguments `n` et `taille` des entiers avec `taille` suffisamment grand pour que le codage de `n` en liste de booléens soit fidèle.

25 . Compléter la nouvelle version de la fonction `recherche` qui utilise la conversion des listes de booléens en entiers :

```
def recherche(but,init):
    taille=len(but[0])
    N=2**taille
    espace=[versEntier(init[0]+init[1])]
    but_int=versEntier(but[0]+but[1])
    stop=False
    while not stop:
        ancien=espace
        espace=espace+successeurs(espace,taille,N)
        espace=tri(espace)
        espace=elim_double(espace)
        ...
        ...
        ...
    return False
```

puis écrire une nouvelle version de la fonction `successeurs` adaptée à cette nouvelle fonction `recherche`.

Annexe

```
def recherche(but, init):
    espace = [init]
    stop = False
    while not stop:
        ancien = espace
        espace = espace + successeurs(espace)
        espace.sort() # permet de trier espace par ordre croissant
        espace = elim_double(espace)
        stop = (ancien==espace)
        if but in espace:
            return True
    return False


def successeurs(L):
    res = []
    for x in L:
        L1 = x[0]
        L2 = x[1]
        res.append( avancer_files(L1, False, L2, False) )
        res.append( avancer_files(L1, False, L2, True) )
        res.append( avancer_files(L1, True, L2, False) )
        res.append( avancer_files(L1, True, L2, True) )
    return res


# dans une liste triée, elim_double enlève les éléments apparaissant plus d'une fois
# exemple : elim_double([1, 1, 2, 3, 3]) renvoie [1, 2, 3]

def elim_double(L):
    # À COMPLÉTER



# exemple d'utilisation
# debut et fin sont des listes composées de deux files de même longueur impaire,
# la première étant prioritaire par rapport à la seconde
debut = [[False]*5, [False]*5]
fin = [[False]*3+[True]*2, [False]*3+[True]*2]
print(recherche(fin,debut))
```