

## Proposition de corrigé

Concours : Concours Commun Mines-Ponts

Année : 2017

Filière : MP - PC - PSI

Épreuve : Informatique

Ceci est une proposition de corrigé des concours de CPGE, réalisée bénévolement par des enseignants de Sciences Industrielles de l'Ingénieur et d'Informatique, membres de l'UPSTI (Union des Professeurs de Sciences et Techniques Industrielles), et publiée sur le site de l'association :

<https://www.upsti.fr/espace-etudiants/annales-de-concours>

### A l'attention des étudiants

Ce document vous apportera des éléments de corrections pour le sujet traité, mais n'est ni un corrigé officiel du concours, ni un corrigé détaillé ou exhaustif de l'épreuve en question.

L'UPSTI ne répondra pas directement aux questions que peuvent soulever ces corrigés : nous vous invitons à vous rapprocher de vos enseignants si vous souhaitez des compléments d'information, et à vous adresser à eux pour nous faire remonter vos éventuelles remarques.

### Licence et Copyright

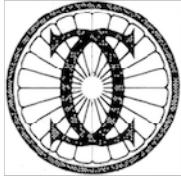
Toute représentation ou reproduction (même partielle) de ce document faite sans l'accord de l'UPSTI est **interdite**. Seuls le téléchargement et la copie privée à usage personnel sont autorisés (protection au titre des droits d'auteur).

En cas de doute, n'hésitez pas à nous contacter à : [corrigesconcours@upsti.fr](mailto:corrigesconcours@upsti.fr).

### Informez-vous !

Retrouvez plus d'information sur les Sciences de l'Ingénieur, l'orientation, les Grandes Ecoles ainsi que sur les Olympiades de Sciences de l'Ingénieur et sur les Sciences de l'Ingénieur au Féminin sur notre site : [www.upsti.fr](http://www.upsti.fr)

L'équipe UPSTI



# ÉTUDE DE TRAFIC ROUTIER

Sujet  
Concours

Corrigé

1h30 ★

Lycée Chaptal - 45 Boulevard des Batignolles - 75008 Paris

**Question 1** Expliquer comment représenter une file de voitures à l'aide d'une liste de booléens. Pour une liste  $L$  de  $n$  éléments, il suffit que chacun des éléments  $L[i]$  avec  $0 \leq i < n$  soit définie par un booléen **False** ou **True**. Le booléen **False** représentant l'absence de voiture dans une case et le booléen **True** la présence de voiture dans la case.

**Question 2** Donner une ou plusieurs instructions Python permettant de définir une liste  $A$  représentant la file de voitures illustrée par la Figure 1(a).

$A=[\text{True}, \text{False}, \text{True}, \text{True}, \text{False}, \text{False}, \text{False}, \text{False}, \text{True}]$

**Question 3** Soit  $L$  une liste représentant une file de longueur  $n$  et  $i$  un entier tel que  $0 \leq i < n$ . Définir 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.

```
1 def occupe(L,i) :
2     return L[i] # On retourne un booléen True ou False
```

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

Il y a  $2^n$  possibilités de files de voitures de longueurs  $n$ . En effet, par case il y a deux possibilités (soit la case est vide soit la case contient une voiture), puisqu'il y  $n$  cases indépendantes, il y a donc  $2^n$  possibilités.

**Question 5** Écrire une fonction `egal(L1,L2)` retournant un booléen permettant de savoir si deux listes  $L1$  et  $L2$  sont égales.

```
1 def egal(L1,L2) :
2     return L1==L2 # On retourne un booléen True ou False
```

**Question 6** Que peut-on dire de la complexité de cette fonction?

La complexité de cette fonction est au pire en  $\text{grandO}(n)$ . En effet, il faut tester un à un les éléments des listes  $L1$  et  $L2$ . Le pire des cas en terme de complexité est que les deux listes soient égales. Cette question me laisse penser que ce qui était attendu par le poseur de sujet était plus quelque chose comme :

```
1 def egal(L1,L2) :
2     if len(L1) != len(L2):
3         return False # On retourne False si les deux listes n'ont pas la même taille
4     else :
5         for i in range(len(L1)):
6             if L1[i]!=L2[i]:
7                 return False # On retourne False dès que deux éléments sont différents
8     return True # Tous les éléments sont identiques
```



**Question 7** Préciser le type de retour de cette fonction.

Je ne comprends pas la question. Il est demandé à ce que ce soit un booléen à la question 5, c'est donc un booléen ...

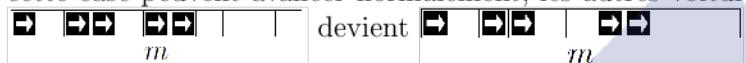
**Question 8** Étant donnée  $A$  la liste définie à la question 2, que renvoie `avancer(avancer(A, False), True)` ?

Pour répondre à cette question, on rappelle que :

$A=[\text{True}, \text{False}, \text{True}, \text{True}, \text{False}, \text{False}, \text{False}, \text{False}, \text{True}]$ , ainsi `avancer(A, False)` renvoie la nouvelle liste  $B$  comme précisée dans le sujet à la Figure ?? à savoir :

$B=[\text{False}, \text{True}, \text{False}, \text{True}, \text{True}, \text{False}, \text{False}, \text{False}, \text{False}]$  De ce fait, la commande `avancer(B, True)` renvoie une nouvelle liste notée  $D$  telle que :

$D=[\text{True}, \text{False}, \text{True}, \text{False}, \text{True}, \text{True}, \text{False}, \text{False}, \text{False}]$

**Question 9** On considère  $L$  une liste et  $m$  l'indice d'une case de cette liste ( $0 \leq m < \text{len}(L)$ ). 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

Définir 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  $L$ .

Il est précisé dans le sujet qu'il ne faut pas modifier la liste  $L$ , il est donc nécessaire d'en faire une copie. Pour cela, on peut utiliser l'extraction `L_copie=L[:]`

```

1 | def avancer_fin(L,m):
2 |     L_copie=L[:]
3 |     L_copie[m]=False # La case m est forcément vide
4 |     for i in range(m+1,len(L_copie)): # On parcourt la sous-liste
5 |         L_copie[i]=L[i-1] # La position du nouvel élément correspond à celle en i-1
6 |     return L_copie

```

ou

```

1 | def avancer_fin(L,m):
2 |     return L[:m]+avancer(L[m:],False)

```

La case d'indice  $m$  dans la nouvelle liste est donc nécessairement inoccupée.

**Question 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 lorsqu'aucune nouvelle voiture n'est introduite.

Définir 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  $L$ .

Il est précisé dans le sujet qu'il ne faut pas modifier la liste  $L$ , il est donc nécessaire d'en faire une copie. Pour cela, on peut utiliser l'extraction `L_copie=L[:]`

```

1 | def avancer_debut(L,b,m):
2 |     L_copie=L[:]
3 |     L_copie[0]=b # La case 0 dépend du paramètre b
4 |     for i in range(1,m+1): # On parcourt la sous-liste
5 |         L_copie[i]=L[i-1] # La position du nouvel élément correspond à celle en i-1

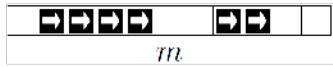
```



6 | `return L_copie`

ou

```
1 | def avancer_debut(L,b,m):
2 |     avancer(L[:m+1],b) + return L[m+1:] +
```

**Question 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(L,b,m)` qui réalise cette étape partielle de déplacement et renvoie le résultat dans une nouvelle liste.

En constatant une petite astuce, cette question ne devient pas très dure. En effet, la fonction `avancer_debut(L,b,m)` est utilisé pour une case d'indice  $m$  **inoccupée**. Il suffit d'utiliser cette fonction mais pour une case inoccupée d'indice  $i$  le plus grand possible tel que  $0 \leq i < m$ . Il faut trouver cet indice  $i$

```
1 | def avancer_debut_bloque(L,b,m):
2 |     i=m-1 # On initialise l'indice le plus grand possible
3 |     while L[i]==True and i>=0: # La condition i$geq0$ permet de vérifier que l'on dépasse la
|         borne gauche de la file
4 |         i=i-1
5 |     if i>=0: # On vérifie que la condition d'arrêt a été vérifiée pour une détection d'une case
|         inoccupée
6 |         return avancer_debut(L,b,i)
7 |     else :
8 |         return L[:] # Si on n'a détecté aucune case inoccupée on retourne une copie de la liste
```

**Question 12** En utilisant le langage Python, définir la fonction `avancer_files(L1,b1,L2,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. Pour cette question, il faut bien avoir compris que la file de voiture  $L1$  avancera quoi qu'il se passe car elle est prioritaire. Pour la file  $L2$ , plusieurs cas sont à considérer

- Si la case d'indice  $m - 1$  de la liste  $L1$  est occupée :
  - ◊ La partie droite (partie basse) de la file  $L2$  pourra avancer et la partie gauche (partie haute) pourra avancer mais sera bloquée à cette case
- Si la case d'indice  $m - 1$  de la liste  $L1$  est inoccupée :
  - ◊ La file de voiture  $L2$  pourra avancer

Ainsi, le codé proposé est le suivant :

```
1 | def avancer_files(L1,b1,L2,b2):
2 |     m=(len(L1)-1)/2 # Valable car longueur impaire
3 |     R1=avancer((L1,m),b1,m)
4 |     R2=avancer\_fin(L2,m)
5 |     if occupe(R1,m):
```



```

6 |     R2=avancer_debut_bloque(R2,b2,m)
7 | else :
8 |     R2=avancer_debut(R2,b2,m)
9 | return [R1,R2]

```

**Question 13** On considère les listes

$$D = [\text{False}, \text{True}, \text{False}, \text{True}, \text{False}], \quad E = [\text{False}, \text{True}, \text{True}, \text{False}, \text{False}]$$

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

Dans aucune des deux files, une voiture n'est insérée. La case du milieu  $m$  à pour indice 2, or dans la file prioritaire  $D$  la case d'indice 1 est occupée. Ainsi, seules les voitures de la file  $E$  à partir de la case d'indice 2 peuvent avancer (c'est-à-dire, celles qui sont sur ou après le carrefour).

$$D = [\text{False}, \text{False}, \text{True}, \text{False}, \text{True}], \quad E = [\text{False}, \text{True}, \text{False}, \text{True}, \text{False}]$$

**Question 14** 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  $L_2$  serait indéfiniment bloquée.

La file  $L_2$  sera indéfiniment bloquée, si on introduit de manière continue une nouvelle voiture à chaque déplacement des voitures de la file  $L_1$  (comme dans la figure 4.a) et que la file  $L_1[0:4] = [\text{True}, \text{True}, \text{True}, \text{True}]$ .

**Question 15** 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.

Pour cette question, le nombre d'étapes minimales correspond à un renouvellement complet de la file  $L_1$  et un déplacement de cinq cases de la file  $L_2$ .

- En 4 étapes, on atteint la situation suivante :  $L_1 = [\text{False}, \text{False}, \text{False}, \text{False}, \text{True}, \text{True}, \text{True}, \text{True}, \text{False}]$  et  $L_2 = [\text{True}, \text{True}, \text{True}, \text{True}, \text{False}, \text{False}, \text{False}, \text{False}, \text{False}]$  ;
- De la 5<sup>me</sup> à la 8<sup>me</sup> étape, la file  $L_2$  avance ;
- Pendant ce temps, on introduit en continu des voitures dans la file  $L_1$  à la 6<sup>me</sup> étape ;
- A la 9<sup>me</sup> étape, on atteint la situation décrite à la figure ??.

**Question 16** Peut-on passer de la configuration 4(a) à la configuration 4(c) ? Justifier votre réponse.  
Impossible, car à l'étape précédente deux voitures seraient sur la même case centrale.

**Question 17** Écrire en langage Python une fonction `elim_double(L)` non récursive, de complexité linéaire en la taille de  $L$ , qui élimine les éléments apparaissant plusieurs fois dans une liste triée  $L$  et renvoie la liste triée obtenue. Par exemple `elim_double([1, 1, 3, 3, 3, 7])` doit renvoyer la liste `[1, 3, 7]`.

Cette question doit être pour vous une question facile, il suffit de faire une boucle sur les éléments de la liste et de vérifier si l'élément courant est égal au précédent puisque la liste est triée. Si c'est le cas, on ne stocke pas cet élément si ce n'est pas le cas, on l'ajoute à une liste  $L\_diff$  contenant les éléments différents

```

1 | def elim_double(L):
2 |     L_diff=[L[0]] # Initialisation de la liste L_diff
3 |     for i in range(1,len(L)): # Cette boucle assure la complexité linéaire
4 |         if L[i]!=L[i-1]:

```



```

5 |     L_diff.append(L[i])
6 | return L_diff

```

**Question 18** Que retourne l'appel suivant `doublons([1,1,2,2,3,3,3,5])` ?

Cette fonction renvoie [1, 2, 3, 5].

**Question 19** Cette fonction est-elle utilisable pour éliminer les éléments apparaissant plusieurs fois dans une liste *non* triée ? Justifier.

Non, cette fonction n'est pas utilisable pour supprimer les « doublons » d'une liste non triée.

**Question 20** La fonction `recherche` donnée 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`. La fonction `recherche` renvoie un booléen qui est `True` si la configuration désirée `but` est atteignable depuis l'état `init`, sinon elle renvoie `False`. `but` est une liste composée de deux sous-listes correspondant à l'état de la première file et de la seconde. `espace` correspond à une liste composée de plusieurs sous-listes, composée elles-même de liste, évoluant au cours de l'algorithme et qui sert notamment à tester les différentes configurations possibles. La fonction `successeurs` renvoie une liste composée de sous-listes, chacune des sous-listes contenant les configurations des `L1` et `L2` suivant le fait que l'on ait ajouté ou non un véhicule à l'entrée d'une file.

**Question 21** Afin d'améliorer l'efficacité du test `if but in espace`, ligne 10 de l'annexe, on propose de le remplacer par `if in1(but, espace)` ou bien par `if in2(but, espace)`, avec `in1` et `in2` deux fonctions définies ci-dessous. On considère que le paramètre `liste` est une liste triée par ordre croissant. Quel est le meilleur choix ? Justifier.

La fonction `in2` est bien entendue la meilleure des deux car elle s'appuie sur l'algorithme de recherche dichotomique d'un élément dans une liste *triée*, sa complexité est donc en  $\mathcal{O}(\log n)$  avec  $n = \text{len}(L)$  tandis que la complexité de la fonction `in1` est linéaire  $\mathcal{O}(n)$ .

**Question 22** 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)` prenant une liste de booléens en paramètre et renvoyant l'entier correspondant. Par exemple, l'appel `versEntier([True, False, False])` renverra 4.

```

1 | def versEntier(L):
2 |     nombre_decimal=0
3 |     n=len(L)
4 |     for i in range(n) :
5 |         nombre_decimal+=L[i]*2**((n-1)-i) # En python, on peut effectuer des calculs directement
       avec des booléens. Dans ce cas True=1 et False=0
6 |     return nombre_decimal

```

ou plus efficacement :

```

1 | def versEntier(L):
2 |     nombre_decimal=0
3 |     for elem in L :
4 |         nombre_decimal=elem+nombre_decimal*2

```

**Question 23** On veut écrire la fonction inverse de `versEntier`, transformant un entier en une liste



de booléens. Que doit être au minimum la valeur de `taille` pour que le codage obtenu soit satisfaisant ?

On suppose que la valeur de taille est suffisante. Quelle condition booléenne faut-il écrire en ligne 4 du code ci-dessous ?

Pour la valeur de `taille` soit suffisante, on doit vérifier la relation suivante :

$$n \leq 2^{\text{taille}} - 1 \implies \text{taille} \geq \left( \frac{\ln n + 1}{\ln 2} \right)$$

La condition à écrire à la ligne 4 est alors `while n!=0 ou while i>=0`.

**Question 24** Montrer qu'un appel à la fonction recherche de l'annexe se termine toujours.

La fonction `recherche` s'arrête à deux conditions :

- La première est que la configuration souhaitée est bien atteignable
- La seconde est atteinte lorsque que les listes `ancien` et `espace` sont les mêmes. Ces listes contiennent des éléments uniques (et qui plus est ordonnées).

Si la première condition est atteinte alors l'appel à la fonction `recherche` est stoppé. Dans le deuxième cas, il faut vérifier que nécessairement à un moment donné ces deux listes deviennent égales. A la question 4, on a vu qu'il y avait  $2^n$  possibilités de files de voitures de longueurs  $n$ . Ainsi, il existe un nombre fini de possibilités pour décrire les files  $L_1$  et  $L_2$  et nécessairement au pire à un moment donné toutes les possibilités auront été testées, de ce fait les listes `ancien` et `espace` seront alors équivalentes.

**Question 25** Compléter la fonction `recherche` pour qu'elle indique le nombre minimum d'étapes à faire pour passer de `init` à `but` lorsque cela est possible. Justifier la réponse.

Il suffit de mettre en place un compteur, ainsi à la ligne 2, on introduit `compteur=0`, il est également indispensable de tester si les listes `init` et `but` sont égales avec :

```
if egal(init, but):
    return compteur
```

après la ligne 9, on écrit alors `compteur+=1` et dans le `return` de la ligne 11, on écrit alors `return True, compteur`

**Question 26** Écrire la requête SQL qui renvoie les identifiants des croisements atteignables en utilisant une seule voie à partir du croisement ayant l'identifiant  $c$ .

```
1 | SELECT id_croisement_fin FROM voie WHERE id_croisement_debut=c
```

**Question 27** Écrire la requête SQL qui renvoie les longitudes et latitudes des croisements atteignables en utilisant une seule voie, à partir du croisement  $c$ .

```
1 | SELECT longitude, latitude FROM croisement JOIN voie ON id_croisement_fin=croisement.id WHERE
  id_croisement_debut=c
```

**Question 28** Que renvoie la requête SQL suivante ?

```
1 | SELECT V2.id_croisement_fin
2 | FROM Voie as V1
3 | JOIN Voie as V2
4 | ON V1.id_croisement_fin = V2.id_croisement_debut
5 | WHERE V1.id_croisement_debut = c
```



Cette requête renvoie les identifiants des croisements atteignables en utilisant exactement deux voies, à partir du croisement  $c$ .



teaching sciences

