

# CORRIGÉ DU DS INFO n°1 - MINES 2017 -

## A. Préliminaires

❑ Q1 – Une file de voiture peut être représentée par une liste de booléen : l'élément d'indice  $i$  contient `True` si et seulement si la case d'indice  $i$  contient une voiture.

❑ Q2 –  $A = [\text{True}, \text{False}, \text{True}, \text{True}] + 6 * [\text{False}] + [\text{True}]$

❑ Q3 –

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

❑ Q4 – Chacune des  $n$  cases contenant ou non une voiture, il y a exactement  $2^n$  files différentes de longueur  $n$ .

❑ Q5 – Qu'attend-on exactement dans cette question ?

On pourrait se contenter de :

```
def egal(L1, L2):
    """ teste si L1 et L2 sont egales """
    return L1 == L2
```

mais l'énoncé semble plutôt attendre cela :

```
def egal(L1, L2):
    "teste si L1 et L2 sont egales"
    if len(L1) != len(L2):
        return False
    for i in range(len(L1)):
        if L1[i] != L2[i]:
            return False
    return True
```

❑ Q6 – La complexité est en  $O(1)$  si les listes ne sont pas de la même longueur et en  $O(n)$  si elles sont toutes deux de longueur  $n$  (le cas le pire étant atteint lorsque les listes sont égales).

❑ Q7 – Cette fonction renvoie un booléen de type `bool`.

*Cette question ressemble à la célèbre devinette : « Quelle est la couleur du cheval blanc d'Henri IV ? »*

## B. Déplacement de voitures dans la file

Cela n'était pas demandé, mais voici comment on pourrait écrire la fonction `avancer` de l'énoncé :

```
def avancer (L, b):
    """ renvoie la liste obtenue apres une etape """
    return [b] + L[:-1]
```

❑ Q8 – `avancer(avancer(A, False), True)` renvoie

`[True, False, True, False, True, True, False, False, False, False, False, False]`.

Cela correspond à la file suivante :



❑ Q9 –

- Version utilisant `avancer` :

```
def avancer_fin(L, m):
    return L[:m] + avancer(L[m:], False)
```

- Version itérative, sans utiliser `avancer` :

```
def avancer_fin(L, m):
    n = len(L)
    L1 = L[:] # Copie de L
    for i in range(n - 1, m, -1):
        L1[i] = L1[i - 1]
    L1[m] = False
    return L1
```

- Version récursive (*inutilement compliqué ici!*) :

```
def avancer_fin(L, m):
    if m == len(L) - 1:
        L1 = L[:] # Copie de L
        L1[-1] = False
        return L1
    else:
        # Déplacement à partir de m + 1
        L1 = avancer_fin(L, m + 1)
        # Déplacement de m
        L1[m + 1] = L1[m]
        L1[m] = False
        return L1
```

#### □ Q10 –

- Version utilisant `avancer` :

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

- Sans utiliser `avancer` :

```
def avancer_debut(L, b, m):
    n = len(L)
    L1 = L[:] # Copie de L
    for i in range(m, 0, -1):
        L1[i] = L1[i - 1]
    L1[0] = b
    return L1
```

- Version récursive :

```
def avancer_debut(L, b, m):
    L1 = L[:] # Copie de L
    if m == 0:
        L1[0] = b
        return L1
    else:
        # Déplacement de m - 1
        L1[m] = L1[m - 1]
        # Déplacement jusqu'à m - 1
        L1 = avancer_debut(L1, b, m - 1)
        return L1
```

#### □ Q11 –

```
def avancer_debut_bloque(L, b, m):
    L1 = L[:] # Copie de L
    for i in range(m - 1, 0, -1):
        if not occupe(L1, i) and occupe(L1, i - 1): # Voiture en i - 1 mais pas en i
            L1[i] = True
            L1[i - 1] = False
    L1[0] = b or L1[0]
    return L1
```

### C. Une étape de simulation à deux files

□ Q12 –

```
def avancer_files(L1, b1, L2, b2):
    m = len(L1) // 2
    R1 = avancer(L1, b1) # Dans tous les cas, L1 avance normalement
    R2 = avancer_fin(L2, m)
    if occupe(R1, m): # une voiture bloque le croisement
        R2 = avancer_debut_bloque(R2, b2, m)
    else:
        R2 = avancer_debut(R2, b2, m)
    return [R1, R2]
```

□ Q13 – L'appel renvoie [[False, False, True, False, True], [False, True, False, True, False]].

### Partie IV. Transitions

□ Q14 – Si toutes les cases de L1 sont toujours occupées, c'est-à-dire si la file est pleine au début et que l'on ajoute une voiture à chaque fois, les voitures de L2 avant le croisement sont indéfiniment bloquées.

□ Q15 – La file représentée par L1 avançant normalement, ses 4 premières voitures doivent passer le croisement (5 étapes) et être remplacées par des cases vides, puis les 4 voitures de L2 qui étaient bloquées vont passer (4 nouvelles étapes) et être remplacées par des cases vides, ces 4 dernières étapes introduisant une voiture dans L1. Cela donne donc au minimum 9 étapes.

□ Q16 – 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.

### D. Atteignabilité

□ Q17 –

```
def elim_double(L):
    "renvoie une liste correspondant a la liste triee L sans repetition."
    if L == []: # par precaution
        return []
    L1 = [L[0]]
    for i in range(1, len(L)):
        if L[i] != L1[-1]:
            L1.append(L[i])
    return L1
```

Autre possibilité pour parcourir une liste, la fonction in :

```
def elim_double(L):
    "renvoie une liste correspondant a la liste triee L sans repetition."
    if L == []: # par precaution
        return []
    L1 = [L[0]]
```

```

for x in L:
    if x != L1[-1]:
        L1.append(x)
return L1

```

❑ Q18 – L'appel de `doublons([1, 1, 2, 2, 3, 3, 3, 5])` renvoie `[1, 2, 3, 5]`. `doublons` est une version récursive de `elim_double`.

❑ Q19 – Cette fonction n'est pas utilisable pour éliminer les éléments apparaissant plusieurs fois dans une liste non triée, puisqu'elle ne compare que deux éléments consécutifs. Par exemple, `doublons([3, 2, 3])` renvoie `[3] + [2] + [3]` soit `[3, 2, 3]`.

❑ Q20 –

- `recherche` renvoie un booléen.
- `successeur` renvoie une liste de listes de 2 listes de même longueur impaire correspondant à des files.
- `espace` pointe sur une liste de listes de 2 listes de même longueur impaire correspondant à des files.
- `but` pointe sur une liste de 2 listes de même longueur impaire correspondant à des files.

❑ Q21 – La recherche par `in1` est de complexité  $O(n)$  (si  $n$  est la longueur de la liste), tandis que celle par `in2` est dichotomique, donc de complexité  $O(\log_2(n)) = O(\ln n)$ . La deuxième est donc préférable.

❑ Q22 – Voici deux versions (directement ou avec schéma de Hörner) :

```

def versEntier(L):
    n = len(L)
    puissance_de_2 = 1 # puissances de 2
    m = 0 # va contenir l'entier
    for i in range(n - 1, -1, -1):
        if L[i]:
            m += puissance_de_2
            puissance_de_2 *= 2
    return m

```

```

def versEntier(L):
    n = len(L)
    m = 0 # va contenir l'entier
    for i in range(n):
        m *= 2
        if L[i]:
            m += 1
    return m

```

❑ Q23 – `taille` doit au moins être égal au nombre de chiffres de l'écriture binaire de  $n$  soit  $\lfloor \log_2 n \rfloor + 1$ . La condition à écrire est  $n > 0$  (ou  $n \neq 0$ ).

❑ Q24 – Par construction de la boucle, la suite des tailles de `espace` est entière strictement croissante. Or le nombre de files possibles est fini donc elle est bornée. Cette suite est donc finie et la boucle se termine.

❑ Q25 –

```

def recherche(but, init):
    espace = [init]
    stop = egal(but, init)
    if stop:
        return 0
    nb_etapes = 0 # On initialise le nombre d'etapes
    while not stop:
        ancien = espace
        espace = espace + successeur(espace)
        espace.sort()
        espace = elim_double(espace)
        stop = egal(ancien, espace)
        nb_etapes += 1 # on incremente le nombre d'etapes
        if but in espace:
            return nb_etapes
    return -1

```

Un invariant d'entrée de boucle est : `espace` contient toutes les configurations possibles, à partir de `init`, en au plus `nb_etapes` étapes, et ne contient pas `but`.

La fonction s'arrête à la première apparition de `but` dans `espace` et renvoie bien le nombre minimal d'étapes pour y arriver (on ne sort de la boucle qu'à la première apparition de `but` dans `espace` s'il y est), et `-1` s'il n'y apparaît pas.

## Partie VI. Base de données

❑ Q26 –

```
SELECT id_croisement_fin FROM Voie WHERE id_croisement_debut = c;
```

❑ Q27 –

```
SELECT longitude, latitude
FROM Croisement JOIN Voie ON Croisement.id = id_croisement_fin
WHERE id_croisement_debut = c;
```

ou bien, beaucoup plus compréhensible mais un peu moins standard :

```
SELECT longitude, latitude FROM Croisement, Voie
WHERE Croisement.id = id_croisement_fin AND id_croisement_debut = c;
```

❑ Q28 – Cette requête renvoie les identifiants des croisement atteignables en utilisant exactement deux voies à partir du croisement ayant l'identifiant  $c$  (donc avec un croisement intermédiaire).

```

*   *   *   *
  *   *   *
    *   *
      *

```