Concours commun Mines-Ponts : corrigé

Jean-Loup Carré

Informatique commune – 2017

Q1. On peut représenter une file de longueur n par une liste L de même longueur. Pour chaque indice i valide, $L[i]$ vaut `True` si la $i^{\text{ème}}$ case de la file est occupée et `False` sinon.

Q2.
```
A = [True, False, True, True] + 6 * [False] + [True]
```

Q3.
```
def occupe(L, i):
    return L[i]
```

Q4. Pour chaque case de la file de longueur n, il y a deux possibilités : elle est libre ou occupée par une voiture. Il y a donc au total 2^n files possibles.

Q5.
```
def egal(L1, L2):
    return L1 == L2
```

Q6. Le test d'égalité `L1 == L2` demande de comparer chaque élément de `L1` à l'élément correspondant de `L2`. La complexité est donc linéaire en la taille des listes.

Q7. Cette fonction renvoie un booléen.

Conseil stratégique

Sun Tsu

La question ressemble à la célèbre devinette « Quelle est la couleur du cheval blanc d'Henri IV ? ». Mais, **ce n'est pas parce que la question est simple que c'est un piège**.

Le poète Du Mu rapporte[1] *que Zhuge Liang, assiégé par le stratège Sima Yi, décida d'ouvrir les portes de sa ville. Sima Yi trouva cela trop facile, cru à un piège et décida de s'enfuir avec ses troupes.*

8. Cette fonction renvoie :
`[True, False, True, False, True, True, False, False, False, False, False]`
Ceci correspond à la file suivante :

9.
```
def avancer_fin(L, m):
    return L[:m] + avancer(L[m:],False)
```

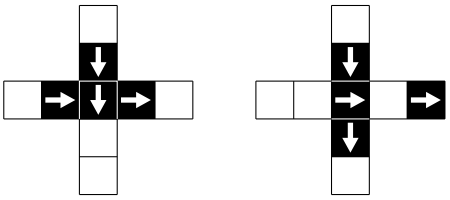
10.
```
def avancer_debut(L, b, m):
    return avancer(L[:m+1], b) + L[m+1:]
```

11.
```
def avancer_debut_bloque(L, b, m):
    L2 = L[:]  # Copier L
    for k in range(m-2, -1, -1):
        if occupe(L2, k+1) == False:
            L2[k+1] = L2[k]
            L2[k] = False
    L2[0] = b or L2[0]
    return L2
```

1. Dans son commentaire de l'*Art de la guerre* de Sun Tsu (chapitre VI). Cette annecdote est aussi rapportée dans *Les 36 stratagèmes* (stratagème 32).

12.
```
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]
```

13. Les files évoluent comme suit :



L'appel renvoie `[[False, False, True, False, True], [False, True, False, True, False]]`.

14. 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.

15. Examinons le temps nécessaire pour que les voitures de la file $L2$ soient dans la position voulue :
    - 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.

16. Pour obtenir la configuration 4(c) en une étape à partir d'une configuration C, il faut que les voitures en position 6 de la configuration 4(c) soient, dans la configuration C toutes les deux en position 5 dans leur liste, donc toutes les deux sur le croisement, ce qui est impossible.

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

17. Voici deux manières différentes de programmer la fonction demandée.

```
def elim_double(L):
    L2=[L[0]]
    for k in range(1,len(L)):
        if L[k] != L[k-1]:
            L2.append(L[k])
    return L2
```

```
def elim_double(L):
    L2=[L[0]]
    for x in L:
        if x != L2[-1]:
            L2.append(x)
    return L2
```

18. `[1, 2, 3, 5]`

Conseil stratégique

Sun Tsu

Dérouler l'exécution de la fonction `doublons` à la main prend du temps, en particulier car cette fonction utilise de manière non-triviale les opérations sur les listes. Une stratégie peut être de deviner ce que veut l'auteur du sujet grâce aux questions précédentes et à une lecture en diagonale du code la fonction ; puis d'en déduire le résultat attendu. Cette stratégie est risquée (l'auteur du sujet pourrait avoir écrit exprès une fonction contenant une erreur) mais permet de gagner du temps.

19. Non, car elle ne compare que les valeurs consécutives d'une liste, elle ne peut pas supprimer un doublon de valeurs si les deux valeurs ne sont pas consécutives.

20.

variable	type
but	liste de listes de booléens
espace	liste de listes de listes de booléens

fonction	type de la valeur de retour
recherche	booléen
successeur	liste de listes de listes de booléens

21. `in1` parcourt la liste élément par élément, d'où une complexité en $\mathcal{O}(n)$ (avec $n =$ `len(liste)`).
`in2` fait une recherche dichotomique, sa complexité est donc (cours) en $\mathcal{O}(\ln(n))$.
Le meilleur choix est donc `in2` (le temps de calcul est plus court).

22.
```
def versEntier(L):
    n = 0
    for b in L:
        n *= 2
        if b:
            n += 1
    return n
```

23.
- `taille` doit valoir au moins le nombre de chiffres de n en base 2, i.e., $\lfloor \log_2(n) \rfloor + 1$ ou $\lceil \log_2(n+1) \rceil$.
- Si `taille` est suffisante, alors il suffit d'écrire dans la condition du `while` : `n != 0` (inutile de tester si on déborde de la liste `res`).

24. Considérons k le nombre de configurations qui ne sont pas dans `espace`. On remarque que k est un variant de boucle (la suite des valeurs de k est une suite d'entiers naturels strictement décroissante), ainsi la boucle ne peut pas être infinie et termine.
La suite des valeurs de k est strictement décroissante car la ligne 6 fait décroître k et la ligne 9 arrête la boucle si k n'a pas strictement décrut (s'il n'a pas strictement décrut, comme les doublons sont supprimés et les listes triées, il y aura égalité à la ligne 9).

25. On ajoute entre les lignes 1 et 2 les lignes suivantes :
```
if egal(init, but):
    return 0
n = 0
```
On ajoute entre les lignes 9 et 10 la ligne : `n = n+1`
Le `return True` de la ligne 11 est remplacé par `return n`
La fonction modifiée est correcte car l'invariant suivant est satisfait : « `espace` contient toutes les configurations atteignables en `n` étapes ou moins ».

Remarque

Le sujet demande d'utiliser `c` comme une constante en SQL. Ceci n'est pas standard et les requêtes suivantes ne marcheront pas si vous les testez sur un ordinateur car `c` sera en fait interprété comme un nom de colonne (et SQL ne trouvera pas la colonne correspondante).

26.
```
SELECT id_croisement_fin FROM croisement WHERE id_croisement_debut = c
```

27.
```
SELECT longitude, lattitude
    FROM croisement JOIN voie ON id_croisement_fin = croisement.id
    WHERE id_croisement_debut = c
```

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

Appendice

Dans l'énoncé, la fonction `avancer` est supposée être déjà programmée. On peut la programmer comme suit.

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