Concours Mines PONTS 2017

Version 1

Par Brahim BAKKAS
CPGE MOULAY ISMAIL
MEKNES
E-Mail: bakkascpge@gmail.com

avr. 30, 2017

Contenu:

1 Mines Ponts 2017, Par brahim BAKKAS **1**

1.1 Partie I. Préliminaires 1

1.2 Partie II. Déplacement de voitures dans la file 2

1.3 Partie III. Une étape de simulation à deux files 3

1.4 Partie IV. Transitions 5

1.5 Partie V. Atteignabilité 5

1.6 Partie VI. Base de données 7

CHAPITRE 1

Mines Ponts 2017, Par brahim BAKKAS

Une proposition de solution pour le concours Mines Ponts 2017 partie informatique

1.1 Partie I. Préliminaires

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 inditèrenciées.

— Q1 – Expliquer comment représenter une file de voitures à l'aide d'une liste de booléens.

pour représenter la file de voiture à l'aide d'une liste, on va mettre True dans la i ème case si une voiture occupe l'emplacement i, sinon False. (cad initiliser une liste avec False si une voitre entre dans la fille on change l'état de la position on True)

— Q2 – Donner une ou plusieurs instructions Python permettant de définir une liste A représentant la file de voitures illustrée par la Figure 1(a).

```
#Q2 Sol1
L= [False]*11
L[0],L[2:4],L[-1]=True,[True,True],True
#Q2 Sol2
L= [False]*11
pos=(0,2,3,-1)
for i in pos:
    L[i]=True
```

— Q3 Soit L une liste représentant une file de longueur n et i un entier tel que 0 <= 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.

```
def occupe(L,i):
    return L[i]==True
```

— Q4 – Combien existe-t-il de files diferentes de longueur n ? Justifier votre réponse

le nombre de solution est $2^n $ filles possible, parce que pour chaque posisition on a deux cas soit occupé ou non avec n emplacement 2 * 2 * 2 * 2 *2 = 2^n

— Q5 – Ecrire une fonction egal(L1, L2) retournant un booléen permettant de savoir si deux listes L1 et L2 sont égales.

```
# Sol1
def egal(L1,L2):
    n=len(L1)
    p=len(L2)
    if n!=p : return False
    for i in range(n):
        if L1[i] != L2[i]:
            return False
    return True
# Sol 2
def egal2(L1,L2):
    return L1==L2
```

— Q6 – Que peut-on dire de la complexité de cette fonction ?

la complexité de cette fonction et O(n)

1.2 Partie 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. On suppose avoir écrit 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.

— Q8 – Etant donnée A la liste définie à la question 2, que renvoie avancer(avancer(A, False),True) ?

```
def avance(L,Bol):
    if Bol ==True:
        return [True]+ L[:-1]
    else:
        return [False]+L[:-1]
# Sol2
def avance2(L,Bol):
    return [Bol]+ L[:-1]
```

— Q9 - On considère L une liste et m l'indice d'une case de cette liste (0 <= m < len(L)). On s'intéresse à une étape partielle ou 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 m devient m 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.

```
# Q9 sol1
def avancer_fin(L, m):
    R = L[:m]+[False]+L[m:-1]
```

```
    return R

# Sol2 avec une boucle pour
#  pour i = n-1 jusuqu'a m   l(n-1)=l(i)
#
def avancer_fin2(L,m):
    R = L[:m]+[False]
    n = len(L)
    for i in range(n-1,m):
        R[i] = L[i-1]
    R[m] = False
    return R
```

— Q10 – Soient L une liste, b un booléen et m l'indice d'une case inoccupée de cette liste. On considère une étape partielle ou 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. 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.

```
#Q10
def avancer_debut(L, b, m):
    return [b]+L[1:m-1]+L[m:]
```

— Q11 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.

```
#Q11
def avancer_debut_bloque(L, b, m):
    i = m-1  # position
    p = m    # position des voitures en stop
    while i>0 and L[i]==True:
        i-=1
        p-=1
    return [b]+L[:i]+L[p:]
```

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.

1.3 Partie III. Une étape de simulation à deux files

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. On suppose avoir écrit 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.

— Q12 – 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

```
def avancer_files(L1, b1, L2, b2):
    """
        L1 liste 1,
        L2 liste 2,
        b1 et b2 des bolean
    """
    R1 = L1.copy()  # R1=L1[:]
    R2 = L2.copy()
    n = len(R1)
    m = n//2
    if occupe(R1,m):                # ou bien R1[m] == True:
        R1 = avancer_debut_bloque(R1,b1,n)
        R2 = avancer_debut(R2,b2,m)
        R2 = avancer_fin(R2,n)
    elif occupe(R2,m):              # ou bien  R2[m] == True:
        R2 = avancer_debut_bloque(R2,b2,n)
        R1 = avancer_debut(R1,b1,m)
        R1 = avancer_fin(R1,m)
    elif occupe(R1,m-1):            #  ou bien R1[m-1]==True :
        R1 = avancer_debut_bloque(R1,b1,n)
        R2 = avancer_debut(R2,b2,m)
        R2 = avancer_fin(R2,m)
    elif occupe(R2,m-1):            #  ou bien  R2[m-1]==True :
        R1 = avancer_debut_bloque(R1,b1,n)
        R2 = avancer_debut_bloque(R2,b2,n)
    else:
        R1 = avancer_debut_bloque(R1,b1,n)
        R2 = avancer_debut_bloque(R2,b2,n)
    return   R1,R2
```

```
# exemple
L1= [True, False, True, True, False, False, True, False, False, False, True]
L2= [False, True, True, True, False, False, True, False, False, False, True]
b1= False
b2 =False
print('Avant :',L1,"\n",L2)
R1,R2= avancer_files(L1,b1,L2,b2)
print('Après :',R1,"\n",R2)
```

```
Avant :
[True, False, True, True, False, False, True, False, False, False, True]
 [False, True, True, True, False, False, True, False, False, False, True]
 Après :
[False, True, False, True, True, False, False, True, False, False, True]
 [False, False, True, True, True, False, False, True, False, False, True]
```

— Q13 – 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) ?

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

1.4 Partie IV. Transitions

— Q14 – En considérant que de nouvelles voitures peuvent être introduites

sur les premières cases des files lors d'une étape de simulation, décrire une situation ou une voiture de la file L2 serait indéfiniment bloquée.

1.5 Partie V. Atteignabilité

Certaines configurations peuvent être néfastes pour la fluidité du trafic. Une fois ces configurations identifi´ees, il est int´eressant de savoir si elles peuvent appara^ıtre. 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 ecrit le code incomplet donné en annexe. Le langage Python sait comparer deux listes de booléens à l'aide de l'opérateur usuel **<**, on peut ainsi utiliser la méthode sort pour trier une liste de listes de booléens.

— Q17 – Ecrire 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].

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

— Q18 – Que retourne l'appel suivant ?

```
doublons([1, 1, 2, 2, 3, 3, 3, 5])
```

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

— **Q19 Cette fonction est-elle utilisable pour éliminer les éléments apparaissant plusieurs fois dans une liste non triée ? Just**
Réponse :Non

— Q20 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.

```
def in1(element,liste):
a = 0
b = len(liste)-1
while a <= b and element >= liste[a]:
    if element == liste[a]:
        return True
    else:
        a = a + 1
return False
```

```
def in2(element,liste):
a = 0
b = len(liste)-1
while a < b:
    pivot = (a+b) // 2 # l'op´erateur // est la division enti`ere
    if liste[pivot] < element:
        a = pivot + 1
    else:
        b = pivot
if element == liste[a]:
    return True
else:
    return False
```

— Q22 – Afin de comparer plus efficacement les files repr´esent´ees par des listes de bool´eens on remarque que ces listes repr´esentent un codage binaire ou True correspond à 1 et False à 0. Ecrire 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.
```

```
def versEntier(L):
    n=len(L)
    v=0
    for i in range(L):
        v+=v*L[i]**(n-1-i)
    return v
```

— Q23 – 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 ?

```
def versFile(n, taille):
res = taille * [False]
i = taille - 1
while n !=0:          # oubien while i !=0:
    if (n % 2) != 0: # % est le reste de la division entière
        res[i] = True
    n = n // 2 # // est la division entièere
    i = i - 1
return res
```

— Q24 – Montrer qu'un appel à la fonction recherche de l'annexe se termine toujours.

```
1   def recherche(but, init):
2   espace = [init]
3   stop = False
4   while not stop:
5   ancien = espace
6   espace = espace + successeurs(espace)
7   espace.sort() # permet de trier espace par ordre croissant
8   espace = elim_double(espace)
9   stop = egal(ancien,espace) # fonction définie `a la question 5
10  if but in espace:
11  return True
12  return False
13
14
15  def successeurs(L):
```

```
16  res = []
17  for x in L:
18  L1 = x[0]
19  L2 = x[1]
20  res.append( avancer_files(L1, False, L2, False) )
21  res.append( avancer_files(L1, False, L2, True) )
22  res.append( avancer_files(L1, True, L2, False) )
23  res.append( avancer_files(L1, True, L2, True) )
24  return res
```

Q25 – 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.

1.6 Partie VI. Base de données

On modélise ici un réseau routier par un ensemble de croisements et de voies reliant ces croisements. Les voies partent d'un croisement et arrivent a un autre croisement. Ainsi, pour modéliser une route à double sens, on utilise deux voies circulant en sens opposés. La base de données du réseau routier est constituée des relations suivantes : * Croisement(id, longitude, latitude) * Voie(id, longueur, id_croisement_debut, id_croisement_fin)

Dans la suite on considère c l'identifiant (id) d'un croisement donné.

— Q26 – Ecrire la requête SQL qui renvoie les identifiants des croisements atteignables en utilisant une seule voie à partir du croisement ayant l'identifiant c.

```
SELESCT id
FROM Croisement,Voie
WHERE Croisement.id,Voie.id_croisement_fin
AND Voie.id_croisement_debut ="c"
```

— Q27 – Ecrire la requête SQL qui renvoie les longitudes et latitudes des croisements atteignables en utilisant une seule voie, à partir du croisement c. .. code : : sql

SELESCT id,longitude, latitude FROM Croisement,Voie WHERE Croisement.id,Voie.id_croisement_fin AND Voie.id_croisement_debut = c

— Q28 – Que renvoie la requête SQL suivante ?

```
SELECT V2.id_croisement_fin
FROM Voie as V1
JOIN Voie as V2
ON V1.id_croisement_fin = V2.id_croisement_debut
WHERE V1.id_croisement_debut = c
```

— modindex
— search