CORRIGÉ DE L'ÉPREUVE Mines-Ponts 2016 D'IPT

Partie I

Question 1 –

- Après le passage où `i=1` : `L` a pour contenu `[2,5,3,1,4]` ;
- Après le passage où `i=2` : `L` a pour contenu `[2,3,5,1,4]` ;
- Après le passage où `i=3` : `L` a pour contenu `[1,2,3,5,4]` ;
- Après le passage où `i=4` : `L` a pour contenu `[1,2,3,4,5]` .

Question 2 – Supposons que pour un certain i compris entre 0 et $n-2$ la propriété $\mathcal{P}_i$: "la liste `L[0:i+1]` est triée par ordre croissant à l'issue de l'itération i" est vérifiée.
Montrons qu'alors $\mathcal{P}_{i+1}$ l'est.
Notons `j0` le dernier `j` (dans l'ordre de déroulement de la boucle interne) pour lequel la condition `0<j and x<L[j-1]` est vérifiée.
Alors, après exécution de cette boucle interne :

- `L[j0+1:i+2]` est égal à la valeur avant cette étape de `L[j0:i+1]` ; donc est trié en vertu de $\mathcal{P}_i$;
- `L[0:j0]` est égal à la valeur avant cette étape de `L[0:j0]` ; donc est trié en vertu de $\mathcal{P}_i$;
- `L[j0]` a pris la valeur `x` , et donc, en raison de la condition d'arrêt de la boucle interne, `L[j0]>L[j0-1]` et `L[j0]<L[j0+1]` .

ces trois points assurent que `L[0:i+2]` est triée.
On a donc montré que $\mathcal{P}_i$ est un invariant de boucle.
Pour montrer que `tri(L)` trie bien la liste `L` , il reste à montrer que :

- Avant le début de la boucle, $\mathcal{P}_0$ est vérifiée ; ce qui est bien le cas car `L[0:1]` comporte un seul élément.
- La sortie de la boucle (après dernier passage pour lequel `i=n-1`) implique que `L` est triée. C'est bien le cas car après ce dernier passage, `L[0:n-1+1]` , c'est à dire `L` , est triée.

Remarque : la terminaison ne posait ici pas de problème car il s'agit d'une boucle for.

Question 3 – Dans le pire des cas, la boucle interne fait i comparaisons (sans compter le `j>0`) et donc on a en tout une complexité (en nombre de comparaisons) :

$$C(n) = 1 + 2 + \cdots + n - 1 = \frac{n(n-1)}{2} = \mathcal{O}(n^2) \ .$$

Dans le meilleur des cas, la boucle interne fait 1 comparaisons et donc on a une complexité linéaire.

Le tri fusion est un tri qui a une complexité dans le pire des cas en $\mathcal{O}(n \ln n)$. Sa complexité dans le meilleur des cas est également $\mathcal{O}(n \ln n)$.

Question 4 – On reprend et on adapte le tri présenté au début :

```
def tri_chaine(L):
    n = len(L)
    for i in range(1,n):
        j = i
        x = L[i][:]
        while 0 < j and x[1] < L[j-1][1]:
            L[j] = L[j-1][:]
            j = j-1
        L[j] = x[:]
```

Question 5 – Dans la table `palu`, aucun attribut ne peut servir de clé primaire car pour chacun de ces attributs, plusieurs lignes peuvet contenir la même valeur.
Le couple `(iso,annee)` peut servir de clé primaire.

Question 6 –

```
SELECT *
FROM palu
WHERE annee=2010 AND deces>1000 ;
```

Question 7 –

```
SELECT P.nom,(P.cas/D.pop)*100000
FROM palu AS P JOIN demographie AS D
     ON P.iso=D.pays AND P.annee=D.periode
WHERE P.annee=2011 ;
```

Question 8 – La requête suivante permet de sélectionner les lignes concernant 2010 et telles que le nombre de cas n'est pas maximum :

```
SELECT *
FROM palu
WHERE annee=2010 AND cas < (SELECT MAX(cas)  FROM palu WHERE annee=2010);
```

Maintenant il suffit d'aller chercher le pays ayant le maximum de cas dans ce nouveau tableau :

```
SELECT nom
FROM (SELECT *
      FROM palu
      WHERE annee=2010 AND cas < (SELECT MAX(cas)  FROM palu WHERE annee=2010) )
WHERE  cas = (SELECT MAX(cas)
              FROM (SELECT *
                    FROM palu
                    WHERE annee=2010 AND cas < (SELECT MAX(cas)
                                                FROM palu WHERE annee=2010)));
```

Remarque : on pouvait sûrement faire plus simple avec d'autres commandes SQL.

Question 9 – La fonction `tri_chaine` est adaptée ; il suffit donc d'exécuter la commande suivante :

```
tri_chaine(deces2010)
```

Partie II

Question 10 – On prend $X(t) = (S(t), I(t), R(t), D(t))$ et pour f la fonction de $\mathbb{R}^4$ dans $\mathbb{R}^4$ définie par :

$$f(S, I, R, D) = (-rSI, rSI - (a+b)I, aI, bI) \ ;$$

où r, a, b sont les constantes introduites dans l'énoncé.

Question 11 –

```
def f(X) :
    return np.array([-r*X[0]*X[1],r*X[0]*X[1]-(a+b)*X[1],a*X[1],b*X[1]])
```

Question 12 – Le résultat obtenu par la première simulation présente des sauts et ne semble pas se stabiliser ; alors que la seconde simulation renvoie des courbes qui indiquent que chacune des variables semble tendre vers une limite.
La mauvaise qualité de la première simulation s'explique par le fait que le pas dans la méthode d'Euler était trop grand.
La seconde simulation a nécessité un temps de calcul plus long.

Question 13 – ligne 7 :

```
def f(X,Itau) :
    return np.array([-r*X[0]*Itau,r*X[0]*Itau-(a+b)*X[1],a*X[1],b*X[1]])
```

ligne 28 :

```
for i in range(N):
    t=t+dt
    if i-p<0 :
        Itau=0.05
    else :
        Itau=XX[i-p][1]
    X=X+dt*f(X,Itau)
    tt.append(t)
    XX.append(X)
```

Question 14 – Il faut remplacer les lignes définissant `Itau` :

```
for i in range(N):
    t=t+dt
    Somme=0 # somme destin\'e \'a approcher l'int\'egrale
    for j in  range(p):
        if i-j<0:
            Somme=Somme+0.05*h(j*dt)
        else :
            Somme=Somme+XX[i-j][1]*h(j*dt)
    Itau=dt*Somme
    X=X+dt*f(X,Itau)
    tt.append(t)
    XX.append(X)
```

Partie III

Question 15 – Cette fonction renvoie une liste de n listes de taille n remplies de 0.

Question 16 –

```
def init(n):
    G=grille(n)
    i=randrange(n)
    j=randrange(n)
    G[i][j]=1
    return G
```

Question 17 –

```
def compte(G):
    n=len(G)
    l=[0,0,0,0]
    for in range(n):
        for j in range(n):
            l[G[i][j]]+=1
    return l
```

Question 18 – Le résultat renvoyé par cette fonction est un booléen.

Question 19 – ligne 12

```
return (G[i][j-1]-1)*(G[i+1][j-1]-1)*(G[i+1][j]-1)*(G[i+1][j+1]-1)*(G[i][j+1]-1)==0
```

ligne 20 :

```
a=(G[i][j-1]-1)*(G[i+1][j-1]-1)*(G[i+1][j]-1)*(G[i+1][j+1]-1)
b=(G[i][j+1]-1)*(G[i-1][j+1]-1)*(G[i-1][j]-1)*(G[i-1][j-1]-1)
return a*b==0
```

Question 20 –

```
def suivant(G,p1,p2):
    n=len(G)
    Gsuivant=grille(n)
    for i in range(n):
        for j in range(n):
            if G[i][j]==3:
                Gsuivant[i][j]= 3
            elif G[i][j]==1:
                a=bernoulli(p1)
                if a==1:
                    Gsuivant[i][j]= 3
                else :
                    Gsuivant[i][j]= 2
            elif G[i][j]==2:
                Gsuivant[i][j]=[2]
            elif G[i][j]==0:
                a=bernoulli(p2)
                if est_exposee(G,i,j) and a==1:
                    Gsuivant[i][j]= 1
                else :
                    Gsuivant[i][j]= 0
    return Gsuivant
```

Question 21 –

```
def simulation(n,p1,p2):
    G=init(n)
    while comte(G)[1]>0 :
        G=suivant(G,p1,p2)
    l=[comte(G)[0]/n**2,0,comte(G)[2]/n**2,comte(G)[3]/n**2]
    return l
```

Question 22 – A la fin de la simulation, x_1 vaut 0 et la proportion de cases qui ont été atteintes vaut $x_2 + x_3$.

Question 23 –

```
def seuil [Lp2,Lxa]:
    N=len(Lp2)
    i=0 ; j=N-1 # on maintient : Lxa[i]<=0.5<=Lxa[j]
    while j-i>1:
        k=(i+j)//2
        if Lxa[k]<0.5 :
            i=k
        else :
            j=k
    return [Lp2[i],Lp2[j]]
```

Question 24 – On ne peut pas supprimer le test ligne 8, sinon on risquerait de vacciner deux fois la même personne ou la personne déjà infectée (et donc pas le bon nombre de personnes en tout).

Question 25 – Cette commande renvoie un tableau de taille 5×5 remplis de 0 sauf une case contenant un 1 (représentant) la personne infectée et 5 cases placées de manière aléatoire contenant des 2 (représentant des personnes vaccinées).