DST Informatique pour tous
PSI et MP
Durée : 2 heures

15 février

La clarté de la présentation et la qualité de la rédaction font partie de l'évaluation. On veillera à utiliser des noms explicites pour les variables et les fonctions introduites. Le candidat pourra clarifier les programmes par l'ajout de commentaires judicieux si nécéssaire.

Dans tout le sujet on supposera avoir importé la bibliothèque `numpy`, sous la forme :

```
import numpy as np
```

Partie I

La Fédération mondiale du jeu de dames, regroupant soixante-dix-sept pays, tient à jour le classement international et accorde les titres de maître et grand maître international. Elle organise également les championnats mondiaux. La base de données de l'association contient des informations administratives sur les concurrents. Pour simplifier le problème, on considère trois tables : JOUEURS, PARTICIPATION et TOURNOIS.

La table JOUEURS contient les attributs suivants :

— id_J : identifiant d'un individu (entier), clé primaire ;
— nom_J : nom du joueur (chaîne de caractères) ;
— prénom_J : prénom du joueur (chaîne de caractères) ;
— adresse_J : adresse du joueur (chaîne de caractères) ;
— email_J : (chaîne de caractères) ;
— naissance_J : année de naissance (entier)
— nationalité_J
— Points_J : points total du joueur (entier).
— Classement_J (entier) :

La table TOURNOIS contient les attributs suivants :

— id_T : identifiant (entier), clé primaire ;
— nom_du_tournois_T : donnée (caratère) ;
— date_T : date du tournoi (entier) ;
— Lieu_T : lieu et adresse du tournoi
— Pays_T
— vainqueur_T : nom du vainqueur

La table PARTICIPATION contient les attributs suivants :

— id_P : identifiant (entier) de la participation, clé primaire ;
— tournoi_P : identifiant du tournoi (entier id_T) ;
— joueur_P : identifiant du joueur (entier id_J) ;
— classement_P : description de l'état du patient (chaîne de caractères).

JOUEUR	TOURNOI	PARTICIPATION
id_J	id_T	id_P
nom_J	nom_du_tournoi_T	tournoi_P
prenom_J	date_T	joueur_P
adresse_J	Lieu_T	classement_P
email_J	vainqueur_T	
naissance_J	Pays_T	
point_J		
Classement_J		
nationalité_J		

1. Écrire une requête SQL permettant d'extraire le nombre de participants aux tournois en France en 2020.
   SELECT COUNT(id_T) FROM (TOURNOI JOIN PARTICIPATION ON id_T=tounoi_P)
   WHERE Pays_T='France'
2. Écrire une requête SQL permettant d'extraire les noms des vainqueurs des tournois en Angleterre par année.
   SELECT vainqueurs_T FROM TOURNOI GROUP BY date_T WHERE Pays_T='Angleterre'
3. Écrire une requête SQL d'extraire le nombre de tournois par pays en 2023.
   SELECT COUNT(id_T) FROM TOURNOI GROUP BY Pays_T WHERE date_T=2023

```
import numpy as np
import numpy.random as rd
#Partie II: question 1:
#initialiser le plateau
def initialisation(n,p):
    assert p<n
    P=np.zeros((2*n,2*n))

    for j in range(p):
        for k in range((j+1)%2,2*n,2):
            P[j,k]=1
        for k in range(j%2,2*n,2):
            P[2*n-j-1,k]=2
    return P
#Question 2:
# u est dans le plateau
def test(T,u):
    i,j=u
    m=len(T)
    if i>=0 and i<m and j>=0 and j<m:
        return True
    return False
#Question 3:
#score joueur 1 et joueur 2
def score(T):
    J1,J2=0,0
    for u in T:
        for j in u:
            if j==1:
                J1+=1
            if j==2:
                J2+=1

    return J1,J2
#Question 4:
# liste pions joueur j
def pion(T,J):
    L=[]
    m=len(T)
    for i in range(m):
        for j in range(m):
            if T[i,j]==J:
                L.append((i,j))
    return L
#Question 5:
def victoire(T,J):
    S=score(T)
    if S[int(2-J)]==0:
        return True
    return False
#Partie III
#Question 1:
def deplacer(T,u:tuple,v:tuple):
    assert T[v[0],v[1]]==0
    J=T[u[0],u[1]]
    T[u[0],u[1]]=0
    T[v[0],v[1]]=J
```

```
#Question 2
# les des cases possible du piont c
def option(T,c:tuple):
    assert test(T,c)
    m=len(T)
    D=[]
    i,j=c
    J=T[i,j]
    if J==1:
        if i<m-1 and j>0 and j<m-1:
            D=[(i+1,j-1),(i+1,j+1)]
        elif i<m-1 and j==0:
            D=[(i+1,j+1)]
        elif i<m-1 and j==m-1:
            D=[(i+1,j-1)]
    if J==2:
        if i>0 and j>0 and j<m-1:
            D=[(i-1,j-1),(i-1,j+1)]
        elif i>0 and j==0:
            D=[(i-1,j+1)]
        elif i>0 and j==m-1:
            D=[(i-1,j-1)]
    return D
# deplacement possible de c

def deplacement(T,c:tuple):
    D=[]
    Dp=option(T, c)
    for u in Dp:
        if T[u[0],u[1]]==0:
            D.append(u)
    return D
#Question 3:
def prendre(T,u):
    i,j=u
    J=T[i,j]

    D={(i+1,j-1):(i+2,j-2),(i+1,j+1):(i+2,j+2),
    (i-1,j-1):(i-2,j-2),(i-1,j+1):(i-2,j+2)}

    D2={}
    for u in D:

        if test(T,D[u]):
                if T[D[u][0],D[u][1]]==0 and T[u[0],u[1]]==3-J:
                    D2[u]=D[u]

    return D2
#Question 4:
def victoire(T,J):
    S=score(T)
    if S[int(2-J)]==0:
        return True
    L=pion(T,J)
    for v in L:
        if len(prendre(T,v)!=0:
                return False
    return True
```

```
#Question 5:
def Prendre_opt(T,u):

    S=1
    D=prendre(T,u)

    if len(D)==0:
        return 0,[[]]
    i,j=u
    J=T[i,j]
    P=[]
    for v in D:
        T2=T.copy()
        T2[v[0],v[1]]=0
        T2[i,j]=0
        T2[D[v][0],D[v][1]]=J
        S1,P1=Prendre_opt(T2,D[v])
        if S1+1==S:
            for u in P1:
               u=[[v,D[v]]]+u
               P.append(u)
        if S1+1>S:

           S=max(S1+1,S)
           P=[]
           for u in P1:
              u=[[v,D[v]]]+u
              P.append(u)



    return int(S),P
#Question 6
def prise(T,p,u):


        l=len(p)
        J=T[u[0],u[1]]
        T1=T.copy()
        T1[u[0],u[1]]=0
        for k in range(l):
            L=p[k]
            i,j=L[0]
            T1[i,j]=0
        i,j=p[l-1][1]
        T1[i,j]=J
        return T1
#Partie IV
#Question 1:
def degG(G):
     DG={}
     for v in G:
          DG[v]=len(G[v])
     return DG
```

```
#Question 2:
def transpose(G):
      D={}
      for v in G:
          for u in G[v]:
              if u in D:
                 D[u].append(v)
              else:
                 D[u]=[v]
      return D
#Question 3:

def hachage(T):
    h=''
    for u in T:
        for v in u:
            h+=str(int(v))
    return int(h)
#Question 4:
def plateau(h,m):
    t=str(h)
    #m=int(np.sqrt(len(t)))

    T=np.zeros((m,m))

    p=len(t)
    t='0'*(m*m-p)+t
    for i in range(m):
        for j in range(m):

            T[i,j]=int(t[i*m+j])
    return T
#Question 5

def config(T,u):
    l=0
    J=T[u[0],u[1]]
    l,P=Prendre_opt(T,u)
    C=[]
    if l>0:

        for p in P:
            T1=prise(T,p,u)
            C.append((hachage(T1),int(3-J)))
        return C,l
    D=deplacement(T,u)

    for v in D:
        T1=T.copy()
        deplacer(T1,u,v)
        C.append((hachage(T1),int(3-J)))
    return C,l
```

```
# 5-a:
 T=np.array([[0., 0., 0., 0., 0., 0.],
        [0., 0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0., 0.],
        [0., 0., 0., 0., 0., 0.],
        [1., 1., 0., 1., 0., 0.],
        [2., 0., 2., 0., 2., 0.]])

config(T,(5,0))
Out[3]: ([(2000100100002020, 1)], 1)
#5-b:
config(T,(5,2))
Out[4]: ([(200000100100200020, 1), (20110000200020, 1)], 1)




#Question 6:

def configuration(T,J):
    if victoire(T,J):
        return []
    L=pion(T,J)
    if len(L)==0:
        return []

    D=[]
    test=0
    for u in L:
        A,l=config(T,u)
        if l>test:
            test=l
            D=A
        elif test==l:
    return D
#Question 7:
# liste adjacence graph bipartie
def graph(T,J):
    m=len(T)
    if victoire(T,J):
        return {}

    L=pion(T,J)
    if len(L)==0:
        return {}

    G={}
    attente=[]
    D=configuration(T,J)
    attente+=D
    G[(hachage(T),J)]=D
    while len(attente)>0 and len(G)<10000:
        t1,J=attente.pop()
        if (t1,J) not in G:
            D=configuration(plateau(t1,m),J)
            if len(D)>0:
              G[(t1,J)]=D
              attente+=D
    return G
```

```
#Question 8:

def cond_victoire(G,J,m):
    W=[]
    for u in G:
        for v in G[u]:
            if victoire(plateau(v[0],m),J):
                W.append(v)
    return W
#Question 9
def attracteur(G,W):
     A=[]
     tG=transpose(G)
     degG={}
     for v in G:
          degG[v]=len(G[v])
     def parcour(u):
           if u not in A:
              A.append(u)
           if u in tG:
              for v in tG[u]:
                   degG[v]-=1
                   if v[1]==1 or degG[v]==0:
                         parcour(v)

     for u in W:
         parcour(u)
     return A
```