CORRIGE DS4 (X ENS 2016)
RESEAUX SOCIAUX

Question 1 :

reseau_A = [5, [[0,1],[0,2],[0,3],[1,2],[2,3]]]

reseau_B = [5, [[0,1],[1,2],[1,3],[2,3],[2,4],[3,4]]]

Question 2 :

```
def creerReseauVide(n) :
    """  Création d'un réseau vide
    n : int nombre d'invidus du reseau """
    return [n,[]]
```

Question 3 :

```
def estUnLienEntre(paire,i,j) :
    """     paire : list avec len(paire) = 2
    i : int
    j : int
    Fonction qui teste si la paire représente le lien entre i et j
    """
    if [i,j] == paire or [j,i] == paire : ## On teste les deux possibilités d'écriture
        return True
    else :
        return False
```

Question 4 :

```
def sontAmis(reseau,i,j) :
    """     reseau : list, represante un reseau social
    i : int,
    j : int,
    Renvoie True s'il existe un lien d'amitié entre i et j
    """

    if [i,j] in reseau[1] or [j,i] in reseau[1] :
        return True
    else :
        return False
```

Ce code est de complexité m, car dans le pire des cas il faut tester tous les liens d'amitié

Remarque : On peut aussi utiliser la fonction estUnLienEntre

Question 5 :

```
def declareAmis(reseau,i,j) :
    """      reseau : list, represante un reseau social
    i : int
    j : int
    Ajoute (si necessaire) au réseau le lien entre i et j
    """
    if not sontAmis(reseau,i,j) :  ## On teste si le lien d'amitié existe déjà
        reseau[1].append([i,j]) ## On ajoute à  reseau[1] (liste des liens) le lien
```

Ce code est de complexité O(m), car dans le pire des cas il faut tester tout les liens d'amitié pour voir si le lien existe (même complexite que la fonction \textbf{sontAmis})

Question 6 :

```
def listeDesAmisDe(reseau,i) :
    """      reseau : list represantant un reseau social
    i : int
    Renvoie la liste des amis de i
    """
    ami = []    ## Initialisation de la liste des amis de i
    for lien in reseau[1] : ## Pour chacun des liens du réseau
        if lien[0] == i  :  ## Le lien peut s'écrire [i,j]
            ami.append(lien[1])
        elif lien[1] == i : ## ou [j,i]
            ami.append(lien[0])
    return ami
```

Ce code est de complexité O(m), car il faut tester tout les liens d'amitié pour voir si le lien existe.

Question 7 :

parent_A = [5,1,1,3,4,5,1,5,5,7]

parent_B = [3,9,0,3,9,4,4,7,1,9]

Les représentants sont 5, 4, 1, et 3 pour la représentation filiale A et 9, 7 et 3 pour la représentation filiale B.

Question 8 :

```
def creerPartitionsEnSingletons(n) :
    """ n :int
    Renvoie le tableau parent où chaque element est son propre représentant"""

    parent=[]
    for elt in range(n) :
        parent.append(elt)
    return parent

    ## autre posibilité en une ligne : return [i for i in range(n)]
```

Question 9 :

```
def representant(parent,i) :
    """      parent : list
    i : int
    Renvoi le représentant de i"""
    ancetre = parent[i]
    while ancetre!=parent[ancetre] : ## Le representant est son propre parent
        ancetre=parent[ancetre]
    return ancetre
```

Complexité : O(n), au pire des cas on a une filiation en ligne exemple : parent=[1,2,3,...,n-1,n-1]

Question 10 :

```
def fusion(parent,i,j) :
    """      parent : list
    i : int
    j : int
    Fusionne les partition contenant i et j"""
    p = representant(parent,i)
    q = representant(parent,j)
    parent[p] = q
```

Question 11 :

fusion (parent, 0, 1)

fusion (parent, 0, 2)

...

fusion (parent, 0, n-1)

Question 12 :

```
def representant(parent,i) :
    """     parent : list
    i : int
    Renvoie le représentant de i"""
    ancetre = parent[i]
    while ancetre!=parent[ancetre] : ## Le representant est son propre parent
        ancetre=parent[ancetre]
    parent[i] = ancetre ## Modification de la question 12
    return ancetre
\end{python}
\end{py}
```

En terme de compléxité, cette modification ne comporte qu'une opération supplémentaire par rapport à la question 10. La compléxité reste la même, on peut dire que c'est gratuit.

Question 13 :

```
def listeDesGroupes(parent) :
    """     parent : list
    Renvoie les groupes de parents"""
    groupe = []
    lrepresentant = []
    ## Droit de modifier parent ?
    for i in range(len(parent)) :
        ancetre = representant(parent,i)
        if ancetre in lrepresentant  :
            if i!=ancetre :
                groupe[lrepresentant.index(ancetre)].append(i)
        else :
            lrepresentant.append(ancetre)
            if ancetre!=i :
                groupe.append([ancetre,i])
            else :
                groupe.append([ancetre])
    return groupe
```

Question 14 :

```
from random import randint
```

```
def coupeMiniumRandomisee(reseau) :
    # Etape 1 :
    P = creerPartitionsEnSingletons(reseau[0])
    # Etape 2 : Aucune action nécessaire
    # Etape 3 :
    lien = reseau[1]
    m = len(lien)# Nombre de lien non marqués
    while len(listeDesGroupes(P))>2 and m!=0 : ## Nombre de groupes au moins 3 et des liens non marqués
        choix = randint(0,m-1) ## Choix du lien au hasard
        [i,j]=lien[choix] ## Valeur de i et de j
        ri,rj= representant(P,i),representant(P,j)
        if ri!=rj :
            fusion(P,i,j)
        m=m-1
        lien = lien[0:choix] + lien[choix+1:]+[lien[choix]]
    ## Etape 4 : Fusion des groupes (pas de critere ?)
    while len(listeDesGroupes(P))>2 :
        groupe = listeDesGroupes(P)
        fusion(P,groupe[0][0],groupe[1][0])
    ## Etape 5 : Return
    return P
```

Question 15 :

```
def tailleCoupe(reseau,parent) :
    nb = 0 ## Initialisation du compteur
    for lien in reseau[1] : # Parcours de tous les liens
        if representant(parent,lien[0])!=representant(parent,lien[1]) : ## Lien entre les deux groupes si représentant différent
            nb = nb+1
    return nb
```

Question 16 :

```
SELECT id2 FROM liens WHERE id1 = x
```

Question 17 :

```
SELECT nom,prenom
FROM individus  JOIN liens
ON individus.id=liens.id1
WHERE liens.id2=x
```

Question 18 :

SELECT distinct id1 from liens JOIN

(SELECT id1 from liens where id2 = x) as ami

ON liens.id2=ami.id1 where liens.id1<>x