X-ENS-ESPCI 2016 : Réseaux sociaux

Objectifs : Le but de ce sujet est de développer des outils numériques sur les réseaux sociaux.

Remarques :

— Le sujet est plutôt abordable, beaucoup de questions ne posent pas de problèmes particuliers en dehors de deux fonctions plus ardues en fin d'épreuve. La difficulté réside plutôt dans la durée de l'épreuve, puisqu'elle ne dure que deux heures.

— L'utilisation des calculatrices **n'est pas autorisée** pour cette épreuve.

— Le langage de programmation est **obligatoirement** `Python`.

A - Programmation en Python

Partie I. Réseaux sociaux

Question 1. *Représentation des deux réseaux.*

```
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. *Fonction* `creerReseauVide(n)`.

```
def creerReseauVide(n):
    return( [n, []] )
```

Question 3. *Fonction* `estUnLienEntre(paire, i, j)`.

```
def estUnLienEntre(paire, i, j):
    return(i in paire and j in paire)
```

Question 4. *Fonction* `sontAmis(reseau, i, j)`.

```
def sontAmis(reseau, i, j):
    for paire in reseau[1] :
        if estUnLienEntre(paire, i, j) :
            return(True)
    return(False)
```

Dans le pire des cas, les deux individus ne sont pas amis, alors $C_{\text{sontAmis()}}(m) = m \times 2 \times 2 = O(m)$.

Question 5. *Procédure* `declareAmis(reseau, i, j)`.

```
def declareAmis(reseau, i, j):
    if not sontAmis(reseau, i, j):
        reseau[1].append([i, j])
```

C'est une procédure, il n'y a donc pas de renvoi.

Dans le pire des cas, ils ne sont pas amis, donc $C_{\text{declareAmis()}}(m) = 1 + C_{\text{sontAmis()}}(m) + 1 = 2 + O(m) = O(m)$.

Question 6. *Fonction* `declareAmis(reseau, i, j)`.

```
def listeDesAmisDe(reseau, i):
    ListeAmis = []
    for paire in reseau[1]:
        if i in paire:
            ListeAmis.append(sum(paire) - i)
    return(ListeAmis)
```

Dans tous les cas, toutes les paires sont examinées, donc $C_{\text{listeDesAmisDe()}}(m) = m \times 2 = O(m)$.

Partie II. Partitions

Question 7. *Représentation des parents.*

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

Les représentants des 4 groupes sont donc 5, 4, 1 et 3.

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

Les représentants des 3 groupes sont donc 9, 7 et 3.

Question 8. *Fonction* `creerPartitionEnSingletons(n)`.

```
def creerPartitionEnSingletons(n):
    return(list(range(n)))
```

Question 9. *Fonction* `representant(parent, i)`.

```
def representant(parent, i):
    while parent[i] != i:
        i = parent[i]
    return(i)

def representant_rec(parent, i): # version recursive
    if parent[i] == i:
        return(i)
    else :
        return(representant_rec(parent, parent[i]))
```

Dans le pire des cas, $C_{\text{representant()}}(n) = n$ tests $+ n$ affectations $= O(n)$.

Exemple : si `parent = [1, 2, 3, 4, 5, 5]`, le pire des cas est pour `i = 0`.

Question 10. *Procédure* `fusion(parent, i, j)`.

```
def fusion(parent, i, j):
    p = representant(parent, i)
    q = representant(parent, j)
    parent[p] = q
```

Question 11. *Proposition de suite de fusions en $O(n^2)$.*

On cherche à fusionner toutes les partitions d'un groupe, mais on se retrouve initialement dans le pire des cas : le groupe possède n singletons.

On cherche le représentant de 0, puis celui de 1 et on fusionne.

On recherche le représentant de 2 et on fusionne avec celui de 0, etc. On va donc se retrouver à faire $n-1$ fusions.

Le problème, c'est qu'en utilisant la procédure `fusion()` de la question 10, on relance à chaque fois la recherche du représentant de 0, qui augmente de 1 à chaque itération. En effet, à chaque fin de `fusion`, on fait `parent[0] = k`.

On fait donc $N = \sum_{j=1}^{n}(j+1)$ opérations, c'est à dire $N = \frac{(n+1)\cdot n}{2} + n = O(n^2)$ opérations.

C'est donc une complexité quadratique de fusions.

Question 12. *Modification fonction* `representant(parent, i)`.

```
def representant(parent, i):
    Lenfants = []
    while parent[i] != i:
        Lenfants.append(i)
        i = parent[i]
    for k in Lenfants:
        parent[k] = i
    return(i)

def representant_rec(parent, i): # version recursive
    if parent[i] == i:
        return(i)
    else :
        r = representant_rec(parent, parent[i])
        parent[i] = r
        return(r)
```

La fonction cumule deux boucles, celle de la recherche du représentant qui est en $O(k)$ si le représentant de `i` nécessite k recherches, et celle de l'affectation du représentant à tous les parents intermédiaires, qui est aussi en $O(k)$.

La complexité passe donc de $O(k)$ à $2\times O(k)$, elle reste donc en $O(k)$ et peut donc être considérée comme "gratuite".

Question 13. *Fonction* `listeDesGroupes(parent)`.

```
def ListeDesGroupes(parent):
    Lgroupes = []
    Lrep = []
    for i in range(len(parent)):
        r = representant(parent, i)
        ir, nr = 0, len(Lrep)
        while ir < nr and Lrep[ir] != r:
            ir += 1
        if ir == nr :
            Lgroupes.append([i])
            Lrep.append(r)
        else :
            Lgroupes[ir].append(i)
    return(Lgroupes)
```

Partie III. Algorithme randomisé pour la coupe minimum

Question 14. *Fonction* `coupeMinimumRandomisee(reseau)`.

```
def coupeMinimumRandomisee(reseau):
    n = reseau[0]
    parent = creerPartitionEnSingletons(n)
    nG = n # nombre de groupes
    nL = len(reseau[1]) # nombre de liens
    while nG >2 and nL > 0:
        nL -= 1
        iL = random.randint(0, nL)
        [i, j] = reseau[1][iL]
        ri = representant(parent, i)
        rj = representant(parent, j)
        if ri != rj:
            # le lien n'est pas deja inclus dans un groupe
            parent[ri] = rj
            nG -= 1
        reseau[1][iL], reseau[1][nL] = reseau[1][nL], reseau[1][iL]
        # on positionne le lien etudie a la fin du reseau pour ne plus le choisir ensuite
    if nG > 2:
        # A la fin de l'etape randomisee, il reste trois groupes au moins
        r0 = representant(parent, 0)
        i = 1
        while nG > 2:
            # On procede alors dans l'ordre a la fusion des groupes
                                                # jusqu'a n'en avoir plus que 2
            ri = represantant(parent, i)
            if r0 != ri:
                parent[r0] = ri
                nG -= 1
            i += 1
    return(parent)
```

Dans le pire des cas :

- Dans le préambule de la fonction, `creerPartitionEnSingleton()` est en $O(n)$ et les autres opérations en coût constant.
- Dans la première boucle `while`, il est effectué m itérations comportant 2 appels à `representant()` et des opérations à coût constant.
- Dans la seconde boucle `while` conditionnée au test `if`, il est effectué $(n-2)$ itérations comportant un appel à `representant()` qui est maintenant en $O(1)$, et des opérations à coût constant.

Finalement, la complexité de l'algorithme est en $O(n) + O(m \cdot 2 \cdot \alpha(n)) + O(n \cdot 1)$.
Donc une complexité en $O(m \cdot \alpha(n) + n)$.

Question 15. *Fonction* `tailleCoupe(reseau, parent)`.

```
def tailleCoupe(reseau, parent):
    nL = 0
    for [i, j] in reseau[1]:
        if representant(parent, i) != representant(parent, j):
            nL += 1
    return(nL)
```

B - Programmation en SQL

Pour cette partie, il peut être utile de consulter la synthèse sur les Bases de Données disponible sur le réseau pédagogique.

Question 16. *Identifiants des amis de l'individu d'identifiant* `x`.

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

Question 17. *(noms, prénoms) des amis de l'individu d'identifiant* `x`.

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

Question 18. *Identifiants des individus amis avec au moins un ami de l'individu d'identifiant* `x`.

```
SELECT DISTINCT b.id2 FROM liens AS a JOIN liens AS b ON a.id2 = b.id1 WHERE a.id1 = x;
```