Algorithmique
Correction Partiel n° 3 (P3)

Info-spé - S3 – Epita

17 décembre 2019 - 9 : 30

***Solution 1* (Représentation et questions... – *2 points*)**

1. La fermeture transitive de G est un graphe :

   a) Connexe

   b) Complet

2. La liste des sommets de G en ordre suffixe de rencontre : $2, 6, 8, 9, 7, 4, 5, 3, 1$

***Solution 2* (Warshall - Union-Find – *4 points*)**

1. Les composantes connexes (ensembles de sommets) :
   - C_1 : $\{0, 3, 4, 8, 12\}$
   - C_2 : $\{1, 7, 9, 10\}$
   - C_3 : $\{2, 5, 11\}$
   - C_4 : $\{6, 13\}$

2. Quels vecteurs pourraient correspondre au résultat ?

	oui	non
P_1	✓	
P_2		✓
P_3	✓	
P_4		✓

***Solution 3* (I want to be tree – *5 points*)**

Spécifications :

La fonction `isTree(`G`)` détermine si le graphe G est un arbre.

Version 1 : l'algo récursif marque les sommets avec le vecteur des pères. Il retourne un couple (nb sommets, booléen = sans cycle)

```
def __isTree(G, s, P):
    nb = 1
    for adj in G.adjlists[s]:
        if P[adj] == None:
            P[adj] = s
            (n, tree) = __isTree(G, adj, P)
            nb += n
            if not tree:
                return (nb, False)
        else:
            if adj != P[s]:
                return (nb, False)
    return (nb, True)
#-----------------------------------------
def isTree(G):
    P = [None] * G.order
    P[0] = -1
    (nb, tree) = __isTree(G, 0, P)
    return tree and nb == G.order
```

Version2 : l'algo récursif marque les sommets avec un simple vecteur de booléens. Du coup le père du sommet courant est passé en plus en paramètre. L'algo ne retourne pas le nombre de sommets rencontrés : obligation de vérifier que tous sont marqués dans l'algo d'appel (moins optimal).

```
def __isTree2(G, s, M, p):
    M[s] = True
    for adj in G.adjlists[s]:
        if not M[adj]:
            if not __isTree(G, adj, M, s):
                return False
        else:
            if adj != p:
                False
    return True
#-----------------------------------------
def isTree2(G):
    M = [False] * G.order
    if not __isTree2(G, 0, M, -1):
        return False
    for i in range(G.order):
        if not M[i]:
            return False
    return True
```

***Solution 4* (Distances et centre – *6 points*)**

1. **Spécifications :** La fonction `eccentricity(`G`, `s`)` calcule l'excentricité de s dans G.

```
def eccentricity(G, src):
    dist = [-1] * G.order
    q = Queue()
    q.enqueue(src)
    dist[src] = 0
    while not q.isempty():
        s = q.dequeue()
        for adj in G.adjlists[s]:
            if dist[adj] == -1:
                dist[adj] = dist[s] + 1
                q.enqueue(adj)
    return dist[s]
```

2. **Spécifications :** La fonction `center(`G`)` retourne le centre du graphe G.

```
def center(G):
    excMin = eccentricity(G, 0)
    L = [0]
    for s in range(1, G.order):
        exc = eccentricity(G, s)
        if exc < excMin:
            (L, excMin) = ([s], exc)
        elif exc == excMin:
            L.append(s)
    return L
```

***Solution 5* (What is this ? – *3 points*)**

1. Le graphe résultat (NG) :

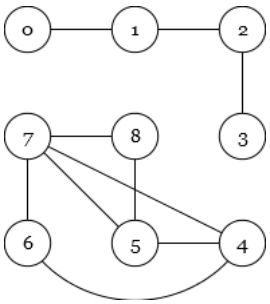

FIGURE 1 – Shuffled Graph

2. Ordre de rencontre des sommets :
   0, 8, 1, 5, 2, 3, 4, 6, 7
3. Combien de composantes connexes lorsque le graphe initial en a k : k