Algorithmics
Correction Final Exam #3 (P3)

Undergraduate 2^{nd} year (S3) – Epita

16 December 2016 - 9 : 30

Solution 1 **(Connections by and large... –** ***4 points*****)**

1. True

   For this problem, we will associate the group of people to a graph where each vertex represents one of those people, the vertices being adjacent when these people know one another. The problem then is to find if there exist graphs whose vertices are all of different degrees. The degrees, however, have to be chosen in the interval $[0, n-1]$. In order for the n vertices to have different degrees, one of them has to be of degree 0 and another one of degree $n-1$, which is impossible.

2. Here is a list of necessary conditions:

   - Each vertex has to have an indegree equal to 2 (each rabbit has two parents), except for two vertices, for which the indegree is equal to 0 (these vertices are the *Adam* and *Ève* of our rabbit group).
   - The graph must be acyclic. Indeed, a rabbit can not have a parent among its own descendants.
   - We must be able to color the vertices of the graph in two colors (male and female), such that every vertex of indegree equal to 2 has a male predecessor and a female predecessor.

3. Only the graph 2 deserves the designation of ***kinship graph***

Solution 2 **(Implementation and questions... –** ***2 points*****)**

1. The transitive closure of G is a:

   a) connected graph

   b) complete graph

2. The Depth-First Search postorder list of vertices of G is: $4, 5, 2, 6, 7, 3, 1$

Solution 3 (B-Trees and Mystery – *3 points*)

1.

	Returned result	*Call number*
(a) `whatIsThis(`B_1`)`	3	21
(b) `whatIsThis(`B_2`)`	13 / 8 (1.625)	22

2. `whatIsThis(`B`)` calculates the average number of keys per node in A (occupancy rate).

Solution 4 (I want to be tree – *5 points*)

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

```python
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).

```python
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 5 **(Diameter –** ***6 points*****)**

Version 1 : la fonction d'appel contient l'init du vecteur de distances. Le parcours largeur (`distance`) retourne le dernier sommet du dernier niveau.

```python
def distance(G, src, dist):
    q = queue.Queue()
    q = queue.enqueue(q, src)
    dist[src] = 0

    while not queue.isEmpty(q):
        s = queue.dequeue(q)
        for adj in G.adjlists[s]:
            if dist[adj] == -1:
                dist[adj] = dist[s] + 1
                q = queue.enqueue(q, adj)

    return s

#----------------------------------------------------------------------------
def diameter(G):
    dist = [-1] * G.order
    s1 = distance(G, 0, dist)
    dist = [-1] * G.order
    s2 = distance(G, s1, dist)
    return dist[s2]
```

Version 2 : le parcours largeur contient l'initialisation du vecteur de distance et retourne le couple (dernier sommet du dernier niveau, sa distance)

```python
def distance(G, src):

    dist = [-1] * G.order
    q = queue.Queue()
    q = queue.enqueue(q, src)
    dist[src] = 0

    while not queue.isEmpty(q):
        s = queue.dequeue(q)
        for adj in G.adjlists[s]:
            if dist[adj] == -1:
                dist[adj] = dist[s] + 1
                q = queue.enqueue(q, adj)

    return (s, dist[s])

#----------------------------------------------------------------------------
def diameter(G):
    (s1, dist) = distance(G, 0)
    (s2, dist2) = distance(G, s1)
    return dist2
```