# Algorithmique
# Correction Contrôle n° 1 (C1)

Info-spé (s3) – Epita

*5 Nov. 2013 - 10 :00*

***Solution 1*** **(Connexions en vrac... –** ***4 points*****)**

1. Vrai

   Pour ce problème, associons au groupe de personnes un graphe dont chaque sommet représente une de ces personnes, les sommets étant adjacents lorsque ces personnes se connaissent. Le problème revient alors à chercher s'il existe des graphes dont les sommets ont des degrés tous différents. Or les degrés sont à choisir dans l'intervalle $[0, n-1]$. Pour que les $n$ sommets aient des degrés différents, un sommet doit donc être de degré 0 et un autre de degré $n-1$, ce qui est impossible.

2. Voici une liste de conditions nécessaires :

   – Chaque sommet doit avoir un degré entrant égal à 2 (chaque lapin a deux parents) à l'exception de deux sommets pour lesquels le degré entrant est nul (ces sommets correspondent aux *Adam* et *Ève* de notre groupe de lapins).

   – Le graphe doit être sans circuit (on dit également acyclique). En effet, un lapin ne peut avoir pour parent l'un de ses descendants.

   – On doit pouvoir colorier les sommets de ce graphe en deux couleurs (mâle et femelle), de façon telle que tout sommet de degré entrant égale à 2 possède un prédécesseur mâle et un prédécesseur femelle.

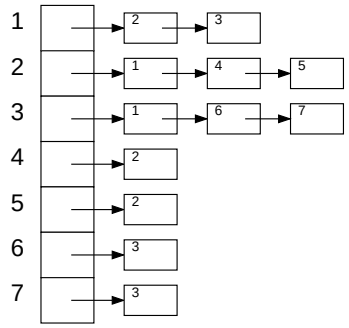
3. Seul le graphe 2 mérite l'appellation de ***graphe de parenté***

***Solution 2*** **(Représentations et question... – *3 points*)**

1. Pour un arbre binaire complet de 7 sommets numérotés hiérarchiquement :
   (a) Sa représentation sous la forme d'une matrice d'adjacence est :

| | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|
| 1 | | $V$ | $V$ | | | | |
| 2 | $V$ | | | $V$ | $V$ | | |
| 3 | $V$ | | | | | $V$ | $V$ |
| 4 | | $V$ | | | | | |
| 5 | | $V$ | | | | | |
| 6 | | | $V$ | | | | |
| 7 | | | $V$ | | | | |

   (b) Sa représentation sous la forme de listes d'adjacence est :




2. Si un graphe $G$ orienté de $n$ sommets et $p$ arcs est implémenté en mémoire sous la forme de listes d'adjacence avec uniquement les listes de successeurs.
   (a) La complexité, au pire, pour calculer le demi-degré extérieur d'un sommet quelconque de $G$ est $d+(x,G)$ (le demi-degré extérieur de $x$ dans $G$), qui dans ce cas là peut-être au pire égal à $n-1$ (pour un 1-graphe).

   (b) La complexité, au pire, pour calculer le demi-degré intérieur d'un sommet quelconque de $G$ est $p$ (le nombre d'arcs du graphe $G$), si ce sommet est le dernier de la dernière liste de successeurs de la représentation du graphe.

***Solution 3*** **(Plus long cycle – *9 points*)**

1. Des arcs pour les cycles :
   (a) Les arcs retours permettent de détecter les cycles.
   (b) Un *cycle élémentaire* sera constitué d'un seul arc retour (et plusieurs arcs couvrants).
   (c) Un arc ($s$, $sadj$) sera un arc retour si $prof[sadj]+1 < prof[s]$ ($sadj$ est un ascendant qui n'est pas le père de $s$).

2. (a) **Spécifications :**
La fonction `plus_long_cycle (t_graph_stat G)` retourne la longueur du plus long cycle dans le graphe $G$ (0 si pas de cycle).

```
algorithme fonction plus_long_cycle : entier
    parametres locaux
        t_graph_stat    G

    variables
        t_vect_entiers    prof
        entier    s, long_max
debut
    pour s <- 1 jusqu'a G.ordre faire
        prof[s] ← -1
    fin pour
    long_max ← 0
    pour s <- 1 jusqu'a G.ordre faire
        si prof[s] = -1 alors
            prof[s] ← 0
            long_max ← max (long_max, plc_rec (G, s, prof))
        fin si
    fin pour
    retourne long_max
fin algorithme fonction plus_long_cycle
```

(b) **Spécifications :**
La fonction `plc_rec (`$G$`, `$s$`, `$prof$`)` lance le parcours profondeur de $G$ à partir du sommet $s$, remplit le vecteur $prof$ (qui sert aussi de marque) avec les profondeur des sommets dans l'arbre couvrant et retourne la longueur du plus long cycle dans le sous-graphe parcouru.

```
algorithme fonction plc_rec : entier
    parametres locaux
        t_graph_stat    G
        entier        s
    parametres globaux
        t_vect_entiers    prof

    variables
        entier     sadj, long_max
debut
    long_max <- 0
    pour sadj <- 1 jusqu'a G.ordre faire
        si G.adj[s,sadj] alors                  /* sadj est un voisin de s */
            si prof[sadj] = -1 alors
                prof[sadj] ← prof[s]+1      /* on marque sadj avec sa profondeur */
                long_max ← max (long_max, plc_rec (G, sadj, prof))
            sinon
                si prof[sadj] + 1 < prof[s] alors       /* (s,sadj) arc retour */
                    long_max ← max (long_max, prof[s]-prof[sadj]+1)
                fin si
            fin si
        fin si
    fin pour
    retourne long_max
fin algorithme fonction plc_rec
```

***Solution 4*** **(Arbre rouge-noir –** ***4 points*****)**

1. L'arbre de la figure 2 est-il complet ? Non car les feuilles sont noires (donc des 2-noeuds).
2. L'algo de test :

```
algorithme fonction estComplet : booleen
      parametres locaux
         t_arn A
    debut
        retourne ( (NUL = A) ou
                    ((A^.rouge ou (A^.fg <> NUL et A^.fg^.rouge et
                                   A^.fd <> NUL et A^.fd^.rouge)) et
                      estComplet(A^.fd) et estComplet(A^.fg)) )
    fin algorithme fonction estComplet
```