Algorithmique
Correction Partiel n° 3 (P3)
(Version profs)

INFO-SPÉ (S3#) – EPITA

5 juin 2017 - 9h

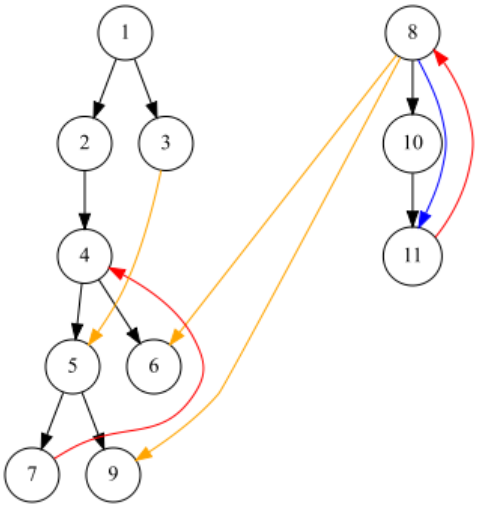
Solution 1 **(Graphes : Sans circuit... –** ***3 points*****)**

1. ***** 1 pt ***** Un graphe sans circuit ne possède pas d'arc retour.

2. ***** 2 pts ***** Les graphes orientés possèdent 4 types d'arcs (u,v) :
   couvrant et avant : si $op[u] < op[v] < os[v] < os[u]$;
   retour : si $op[v] < op[u] < os[u] < os[v]$;
   croisés : si $os[v] < op[u]$
   Dans le cas des arcs couvrants, avants et croisés on a $os[v] < os[u]$. Comme nous sommes dans le cas d'un graphe sans circuit, il n'y a pas d'arc retour. La propriété est donc démontrée.

Solution 2 **(Dans les profondeurs de la forêt couvrante –** ***4 points*****)**

1. ***** 1 + 1 pts ***** *Forêt couvrante et arcs supplémentaires pour le parcours en profondeur, depuis le sommet 1, du graphe de la figure 1 :*



2. ***** 1 + 1 pts ***** *Vecteurs des pères et ordres suffixes :*

	1	2	3	4	5	6	7	8	9	10	11
père	-1	1	1	2	4	4	5	-1	5	8	10
suffixe	8	6	7	5	3	4	1	11	2	10	9

***Solution 3* (Composantes – *4 points*)**

Spécifications :
La fonction `components(G)` retourne le couple (k, cc) où k est le nombre de composantes connexes du graphe non orienté G, et cc le vecteur des composantes.

```
        def __components(G, s, cc, no):
            cc[s] = no
            for adj in G.adjLists[s]:
                if cc[adj] == 0:
                    __components(G, adj, cc, no)

        def components(G):
            cc = [0] * G.order
            k = 0
            for s in range(G.order):
                if cc[s] == 0:
                    k += 1
                    __components(G, s, cc, k)
            return (k, cc)
```

***Solution 4* (Chemin – *6 points*)**

1. ***** 0,5 *****Le parcours largeur donnera un des chemins les plus courts.

2. **Spécifications :**
   La fonction `path(`G`, `src`, `dst`)` recherche un chemin, le plus court possible, à partir du sommet src jusqu'au sommet dst dans le graphe orienté G. Elle retourne la liste des sommets constituant le chemin (dans l'ordre), une liste vide s'il n'existe pas de chemin.

   ***** 5.5 *****

```
        def __pathBFS(G, src, dst, p):
            q = Queue()
            q = enqueue(src, q)
            p[src] = -1
            while not isEmpty(q):
                s = dequeue(q)
                if s == dst:
                    return True
                for adj in G.adjLists[s]:
                    if p[adj] == None:
                        p[adj] = s
#                         if adj == dst:      # if here, does not work
#                             return True     # when src == dst!
                        q = enqueue(adj, q)
            return False

        def pathBFS(G, src, dst):
            p = [None] * G.order
            L = []
            if __pathBFS(G, src, dst, p):
                while dst != -1:
                    L.insert(0, dst)
                    dst = p[dst]
            return L
```

Solution 5 **(Nœuds rouges –** ***3 points*****)**

Spécifications :

La fonction `nbRed(`A`)` retourne le nombre de nœuds rouges que contiendrait un arbre bicolore équivalent à l'arbre 2-3-4 A.

```
        def __nbRed(B):
            n = B.nbKeys - 1
            if B.children != []:
                for i in range(B.nbKeys):
                    n += __nbRed(B.children[i])
            return n

        def nbRed(B):
            if B == None:
                return 0
            else:
                return __nbRed(B)


        # without "call" function

        def nbRed(B):
            if B == None:
                return 0
            n = B.nbKeys - 1
            if B.children != []:
                for i in range(B.nbKeys):
                    n += nbRed(B.children[i])
            return n
```