Algorithmique
Correction Partiel n° 3 (P3)

Info-spé (S3) – Epita

18 décembre 2018 - 9 : 30

Solution 1 **(Warshall - Union-Find – *3 points*)**

1. Les composantes connexes (ensembles de sommets) :
   - $C_1 : \{0, 2, 5, 6\}$
   - $C_2 : \{1, 3, 7, 8\}$
   - $C_2 : \{4, 9\}$
2. Quels vecteurs pourraient correspondre au résultat ?

☐ P_1 ☑ P_2 ☑ P_3 ☐ P_4

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

Forêt couvrante et arcs supplémentaires pour le parcours en profondeur du graphe de la figure 1 :

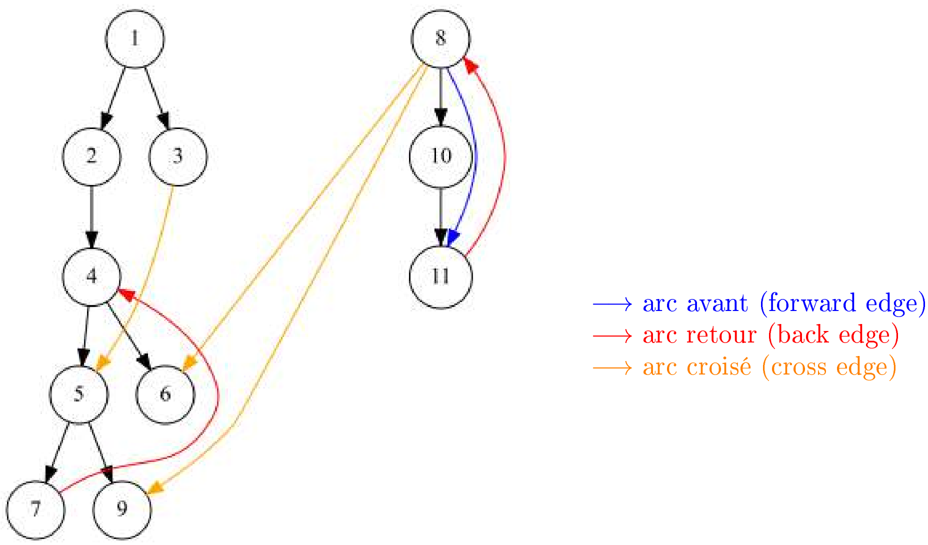

Figure 1 – DFS : Forêt couvrante

Solution 3 **(Composantes –** ***3 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.

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

# with a BFS
def __components_bfs(G, s, cc, no):
    q = queue.Queue()
    cc[s] = no
    q.enqueue(s)
    while not q.isempty():
        s = q.dequeue()
        for adj in G.adjlists[s]:
            if cc[adj] == 0:
                cc[adj] = no
                q.enqueue(adj)

# call
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 **(Diamètre –** ***5 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.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 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 distance2(G, src):
    dist = [-1] * G.order
    q = queue.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 (s, dist[s])

#-------------------------------------------------------
def diameter2(G):
    (s, dist) = distance2(G, 0)
    (s2, dist2) = distance2(G, s)
    return dist2
```

Solution 5 (Euler – *6 points*)

Spécifications :

La fonction `Euler(G)` vérifie si le graphe simple G est eulérien.

```python
def __isEulerian(G, s, M):
    """
    returns (nb, odd) = (nb met vertices, nb odd vertices)
    """
    M[s] = True
    nb = 1
    odd = len(G.adjlists[s]) % 2
    for adj in G.adjlists[s]:
        if not M[adj]:
            (n, o) = __isEulerian(G, adj, M)
            nb += n
            odd += o
            if odd > 2:
                return (nb, odd)
    return (nb, odd)

def isEulerian(G):
    M = [False] * G.order
    (nb, odd) = __isEulerian(G, 0, M)
    return (nb == G.order) and (odd < 3)
```

Remarque : Un graphe ne peut pas avoir 1 seul sommet de degré impair !

Solution 6 (What is this ? – *3 points*)

1. Résultat retourné par `what`(G_4) :

   `lc` = $\boxed{5}$

	0	1	2	3	4	5	6	7	8	9	10	11	12	13
d	0	0	2	1	0	1	3	4	2	1	3	2	3	4

2. d représente :

   pour chaque sommet sa profondeur dans l'arbre couvrant.

3. lc représente :

   la longueur d'un plus long cycle trouvé **lors de ce parcours**, contenant un seul arc retour (ne donne pas forcément le plus long cycle élémentaire du graphe !)