

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) :

- $C_1 : \{0, 2, 5, 6\}$
- $C_2 : \{1, 3, 7, 8\}$
- $C_3 : \{4, 9\}$

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 :

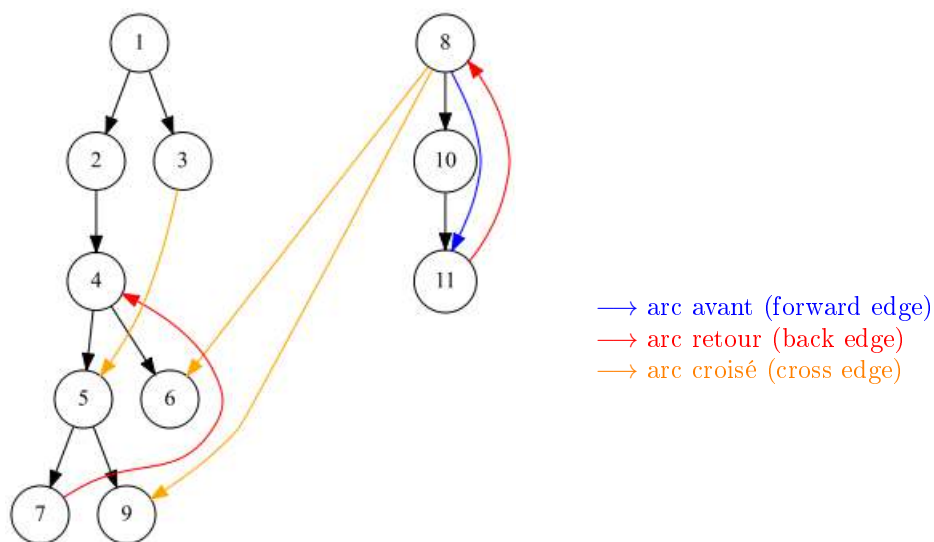


FIGURE 1 – DFS : Forêt couvrante

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.

```
1 # with a DFS
2 def __components(G, s, cc, no):
3     cc[s] = no
4     for adj in G.adjLists[s]:
5         if cc[adj] == 0:
6             __components(G, adj, cc, no)
7
8 # with a BFS
9 def __components_bfs(G, s, cc, no):
10    q = queue.Queue()
11    cc[s] = no
12    q.enqueue(s)
13    while not q.isempty():
14        s = q.dequeue()
15        for adj in G.adjlists[s]:
16            if cc[adj] == 0:
17                cc[adj] = no
18                q.enqueue(adj)
19
20 # call
21 def components(G):
22    cc = [0] * G.order
23    k = 0
24    for s in range(G.order):
25        if cc[s] == 0:
26            k += 1
27            __components(G, s, cc, k)
28    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.

```
1 def distance(G, src, dist):
2     q = queue.Queue()
3     q.enqueue(src)
4     dist[src] = 0
5
6     while not q.isempty():
7         s = q.dequeue()
8         for adj in G.adjlists[s]:
9             if dist[adj] == -1:
10                dist[adj] = dist[s] + 1
11                q.enqueue(adj)
12
13     return s
14
15 #
16 def diameter(G):
17     dist = [-1] * G.order
18     s1 = distance(G, 0, dist)
19     dist = [-1] * G.order
20     s2 = distance(G, s1, dist)
21     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)

```
1 def distance2(G, src):
2     dist = [-1] * G.order
3     q = queue.Queue()
4     q.enqueue(src)
5     dist[src] = 0
6     while not q.isempty():
7         s = q.dequeue()
8         for adj in G.adjlists[s]:
9             if dist[adj] == -1:
10                dist[adj] = dist[s] + 1
11                q.enqueue(adj)
12     return (s, dist[s])
13
14 #
15 def diameter2(G):
16     (s, dist) = distance2(G, 0)
17     (s2, dist2) = distance2(G, s)
18     return dist2
```

Solution 5 (Euler – 6 points)

Spécifications :

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

```

1  def __isEulerian(G, s, M):
2      """
3      returns (nb, odd) = (nb met vertices, nb odd vertices)
4      """
5      M[s] = True
6      nb = 1
7      odd = len(G.adjlists[s]) % 2
8      for adj in G.adjlists[s]:
9          if not M[adj]:
10             (n, o) = __isEulerian(G, adj, M)
11             nb += n
12             odd += o
13             if odd > 2:
14                 return (nb, odd)
15         return (nb, odd)
16
17  def isEulerian(G):
18      M = [False] * G.order
19      (nb, odd) = __isEulerian(G, 0, M)
20      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(G4)` :

$$lc = \boxed{5}$$

	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!)