

# Algorithmique

## Correction Partiel n° 3 (P3)

INFO-SPÉ - S3# - EPITA

14 mai 2019

*Solution 1* (Graphes : dessiner c'est gagner - 2 points)

La forêt couvrante associée au parcours en profondeur du graphe G orienté est celui de la figure 1

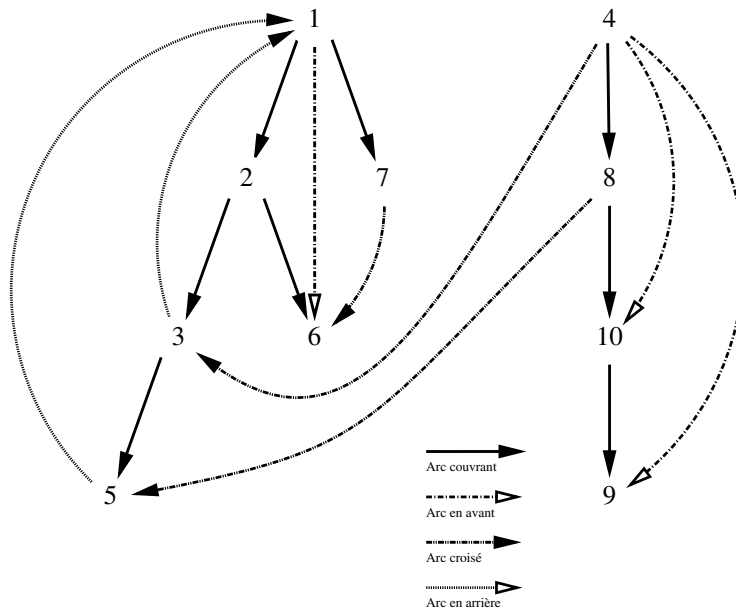


FIGURE 1 – Forêt couvrante du parcours profondeur

**Solution 2 (Union-Find – 3 points)**

1. Nombre de sommets pour chaque composante :  
 $C_1 : 4$        $C_2 : 6$        $C_3 : 4$
2. Arêtes à ajouter : deux arêtes parmi  $5 - 8$   $8 - 12$   $5 - 12$  par exemple...
3. Parmi les chaînes suivantes, celles qui ne peuvent pas exister dans  $G$  :  
 $\square 3 \leftrightarrow 7$        $\boxtimes 11 \leftrightarrow 6$        $\boxtimes 0 \leftrightarrow 13$        $\square 4 \leftrightarrow 9$

**Solution 3 (Graphes bipartis (Bipartite graph) – 5 points)**

Une autre manière de voir le graphe biparti : ses sommets peuvent être "colorés" en deux couleurs de sorte que deux sommets de même couleur ne sont pas adjacents.

**Principe**

Pour tester si un graphe est biparti, il suffit de faire un parcours en largeur ou en profondeur en vérifiant que pour chaque arête empruntée, le sommet de départ n'est pas dans le même ensemble que le sommet d'arrivée. Il faut utiliser un système de marquage à deux valeurs (-1,1) afin de distinguer chaque ensemble.

Le parcours est ici fait en largeur. Dès qu'un sommet non marqué est trouvé, il prend la marque opposée de celle de son père. Si un sommet déjà marqué a la même marque que son prédécesseur le parcours s'arrête (le graphe n'est pas biparti).

Si aucune arête ne relie deux sommets de même marque (le parcours n'a pas été interrompu), le graphe est biparti.

**Spécifications :**

La fonction `bipartite(G)` indique si le graphe non orienté  $G$  est biparti.

```

1 def __bipartiteBFS(G, s, Set):
2     q = queue.Queue()
3     q.enqueue(s)
4     Set[s] = 1
5     while not q.isempty():
6         s = q.dequeue()
7         for adj in G.adjlists[s]:
8             if Set[adj] == 0:
9                 Set[adj] = -Set[s]
10                q.enqueue(adj)
11            else:
12                if Set[adj] == Set[s]:
13                    return False
14        return True
15
16 #
17
18 def bipartiteBFS(G):
19     Set = [0] * G.order
20     for s in range(G.order):
21         if Set[s] == 0:
22             if not __bipartite(G, s, Set):
23                 return False
24     return True

```

```

1 def __bipartite(G, s, Set):
2     for adj in G.adjlists[s]:
3         if Set[adj] == 0:
4             Set[adj] = -Set[s]
5             if not __bipartite(G, adj, Set):
6                 return False
7         else:
8             if Set[adj] == Set[s]:
9                 return False
10    return True
11
12
13
14
15 #
16
17 def bipartite(G):
18     Set = [0] * G.order
19     for s in range(G.order):
20         if Set[s] == 0:
21             Set[s] = 1
22             if not __bipartite(G, s, Set):
23                 return False
24     return True

```

**Solution 4** (Mangez des crêpes – 8 points)

1. Le graphe représentant la recette :

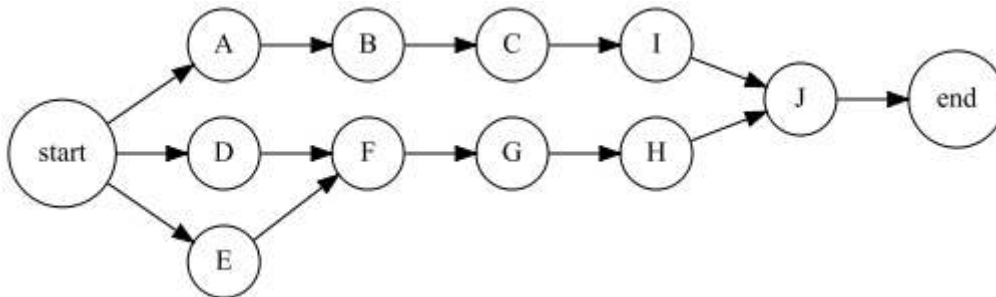


FIGURE 2 – Crêpe à la banane flambée!

2. Le cuisinier est tout seul en cuisine :

- (a) Une solution de tri topologique : *debut* - D - A - E - B - F - C - G - I - H - J - *fin*.
- (b) **Spécifications** : La fonction `tri_topo` ( $G$ ) retourne une solution de tri topologique pour le graphe  $G$  sans circuit, dont tous les sommets sont atteignables depuis le sommet 0.

```
1 def dfsSuff(G, s, M, L):
2     M[s] = True
3     for adj in G.adjlists[s]:
4         if not M[adj]:
5             dfsSuff(G, adj, M, L)
6     L.insert(0, s)
7
8 def topologicalOrder(G):
9     M = [False] * G.order
10    L = []
11    dfsSuff(G, 0, M, L)
12    return L
```

- (c) **Spécifications** : La fonction `is_tri_topo` ( $G, L$ ) vérifie si  $L$  peut être une solution de tri topologique pour le graphe  $G$  sans circuit. La liste  $L$  peut être "détruite"...

```
1 def testTopologicalOrder(G, L):
2     M = [False]*G.order
3     while L != []:
4         s = L.pop()
5         for adj in G.adjlists[s]:
6             if not M[adj]:
7                 return False
8         M[s] = True
9     return True
```

**Solution 5 (What does it do? – 4 points)**

1. Résultat retourné par  $\text{build}(G_3)$  (vecteur des demi-degrés intérieurs des sommets de  $G_3$ ) :

	0	1	2	3	4	5	6	7	8
v	1	2	1	1	0	3	2	1	2

2. La fonction `what` :

- (a) `what( $G_3$ )` retourne :  
la liste  $[4, 3, 7, 6, 0, 1, 2, 5, 8]$
- (b) `what( $G$ )` représente une solution de tri topologique pour  $G$ .
- (c) Propriété de  $G$  pour que `what( $G$ )` ne "plante" pas?

$G$  doit être sans circuit.