

# Algorithmique

## Correction Partiel n° 3 (P3)

INFO-SPÉ - S3 – EPITA

5 janvier 2021 - 9 : 30

**Solution 1** (Dans les profondeurs de la forêt couvrante – 3 points)

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

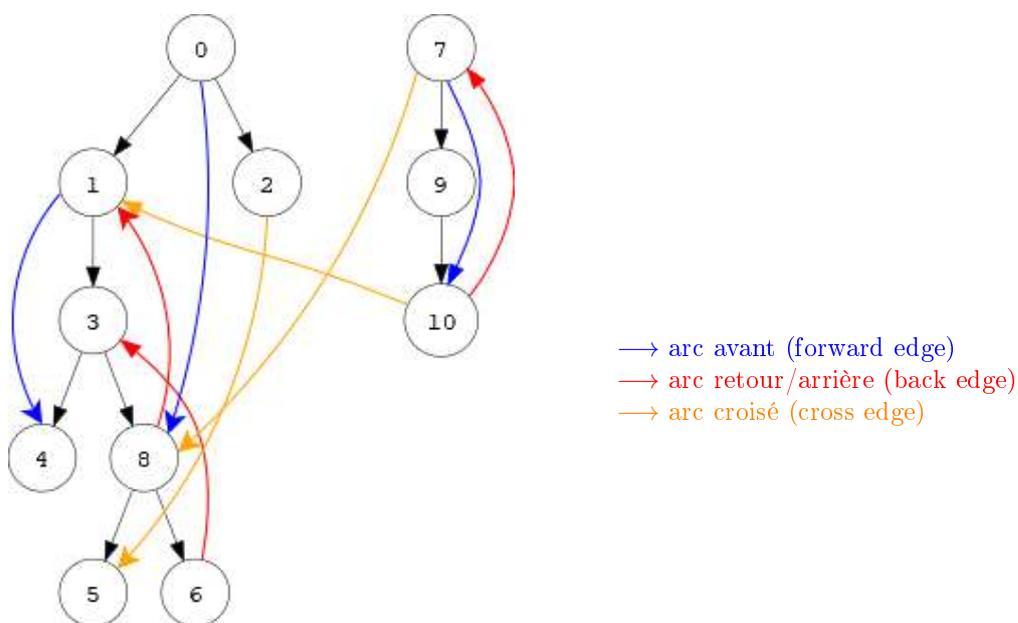


FIGURE 1 – DFS : Forêt couvrante

2. Ordres de rencontre en préfixe **pref** et suffixe **suff** :

|             | 0  | 1  | 2  | 3  | 4 | 5 | 6  | 7  | 8  | 9  | 10 |
|-------------|----|----|----|----|---|---|----|----|----|----|----|
| <b>pref</b> | 1  | 2  | 14 | 3  | 4 | 7 | 9  | 17 | 6  | 18 | 19 |
| <b>suff</b> | 16 | 13 | 15 | 12 | 5 | 8 | 10 | 22 | 11 | 21 | 20 |

**Solution 2 (Union-Find – 4 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  $G$  :  
  $3 \leftrightarrow 7$         $11 \leftrightarrow 6$         $0 \leftrightarrow 13$         $4 \leftrightarrow 9$
4. Vecteur  $p$  après ajout de l'arête 7-4 :

|     |   |   |   |   |    |    |   |   |     |    |    |    |    |    |
|-----|---|---|---|---|----|----|---|---|-----|----|----|----|----|----|
| $p$ | 0 | 1 | 2 | 3 | 4  | 5  | 6 | 7 | 8   | 9  | 10 | 11 | 12 | 13 |
|     | 5 | 8 | 5 | 8 | 12 | -4 | 5 | 8 | -10 | 12 | 8  | 12 | 8  | 8  |

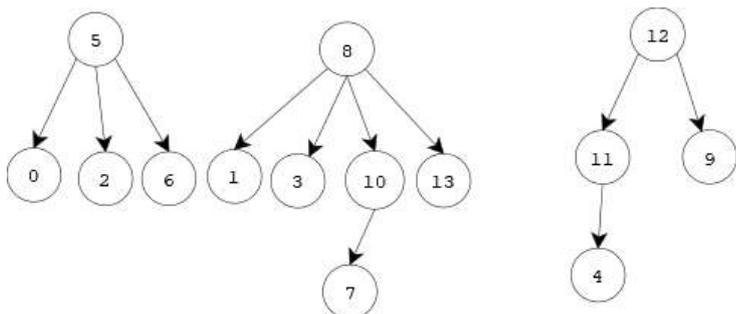


FIGURE 2 – before

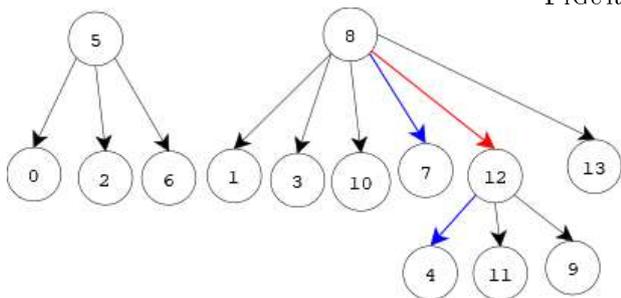


FIGURE 3 – after edge 7 – 4 added

**Solution 3 (Distance au départ – 5 points)**

**Spécifications :** `dist_range(G, src, dmin, dmax)` retourne la liste des sommets à une distance comprise entre  $dmin$  et  $dmax$  du sommet  $src$  dans  $G$  graphe orienté (avec  $0 < dmin \leq dmax$ ).

```

1 def dist_range(G, src, dmin, dmax):
2     dist = [None] * G.order
3     q = queue.Queue()
4     q.enqueue(src)
5     dist[src] = 0
6     L = []
7     while not q.isempty():
8         x = q.dequeue()
9         if dist[x] >= dmin:
10            L.append(x)
11        if dist[x] < dmax:
12            for y in G.adjlists[x]:
13                if dist[y] == None:
14                    dist[y] = dist[x] + 1
15                    q.enqueue(y)
16    return L

```

**Solution 4 (Get cycle – 5 points)**

**Spécifications :**

la fonction `get_cycle(G)` retourne un cycle du graphe non orienté  $G$ , la liste vide si le graphe est acyclique.

**Version 1 :** utilisation de vecteur des pères

```
1 def __get_cycle(G, x, parent):
2     """
3     DFS on G from x, interrupted at first back edge found
4     parent: vertices marked with their parent
5     return first back edge found (x, y) or None
6     """
7     for y in G.adjlists[x]:
8         if parent[y] == None:
9             parent[y] = x
10            get = __get_cycle(G, y, parent)
11            if get != None:
12                return get
13        else:
14            if y != parent[x]:
15                return (x, y)
16    return None
17
18 def get_cycle(G):
19     parent = [None] * G.order
20     s = 0
21     get = None
22     while s < G.order and get == None:
23         if parent[s] == None:
24             parent[s] = -1
25             get = __get_cycle(G, s, parent)
26         s += 1
27     L = []
28     if get != None:
29         (x, y) = get
30         L = [x]
31         while x != y:
32             x = parent[x]
33             L.append(x)
34         L.append(L[0])
35     return L
```

**Version 2 :** la fonction récursive construit le cycle à la remontée  
De nombreuses manières de faire. La difficulté : ne plus ajouter de sommets lorsque le cycle est complet.

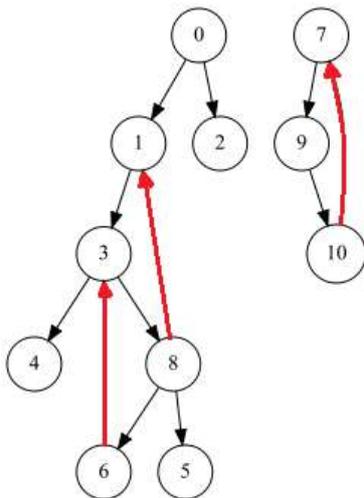
```

1 def __get_cycle2(G, x, M, p):
2     """
3     DFS on G from x
4     M: mark vector (boolean)
5     p: x's parent
6     return (cycle, done):
7     - cycle = the vertices of the first cycle found, [] if no cycle
8     - done: boolean: is the cycle completed?
9     """
10    M[x] = True
11    for y in G.adjlists[x]:
12        if not M[y]:
13            (cycle, done) = __get_cycle2(G, y, M, x)
14            if cycle:
15                if done:
16                    return (cycle, True)
17                if cycle[0] != y:
18                    cycle.append(y)
19                return (cycle, cycle[0] == y)
20            else:
21                if y != p:
22                    return ([y], False)
23    return ([], False)
24
25 def get_cycle_2(G):
26    M = [False] * G.order
27    for s in range(G.order):
28        if not M[s]:
29            cycle, done = __get_cycle2(G, s, M, -1)
30            if cycle:
31                return cycle + [cycle[0]]
32    return []

```

**Solution 5 (What is this? – 3 points)**

1. Le graphe résultat ( $NG$ ) :



2. Pendant le parcours, pour chaque sommet  $s$  :

(a) Que représente  $D[s]$  ?

$D[s]$  est à `None` si le sommet  $s$  n'a pas encore été rencontré (sert de marque). Sinon, c'est la profondeur de  $s$  dans la forêt couvrante du parcours profondeur.

(b) Que représente  $P[s]$  ?

$P[s]$  indique si  $s$  a été rencontré en suffixe.