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 :*

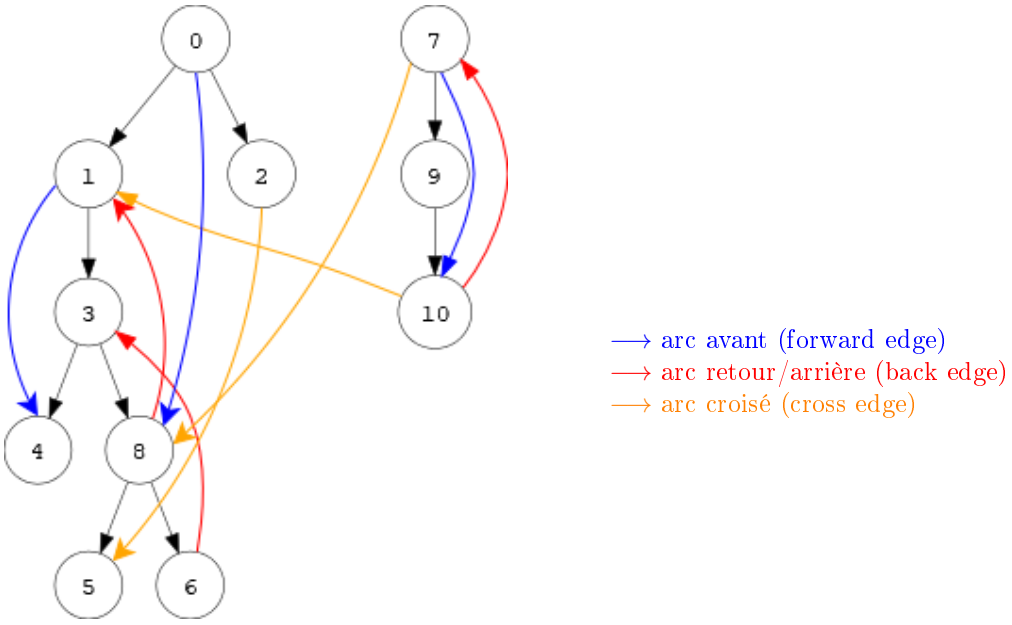


FIGURE 1 – DFS : Forêt couvrante

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

	0	1	2	3	4	5	6	7	8	9	10
pref	1	2	14	3	4	7	9	17	6	18	19
suff	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 ⇜⇝ 7 ☒ **11 ⇜⇝ 6** ☒ **0 ⇜⇝ 13** ☐ 4 ⇜⇝ 9
4. Vecteur p après ajout de l'arête 7–4 :

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

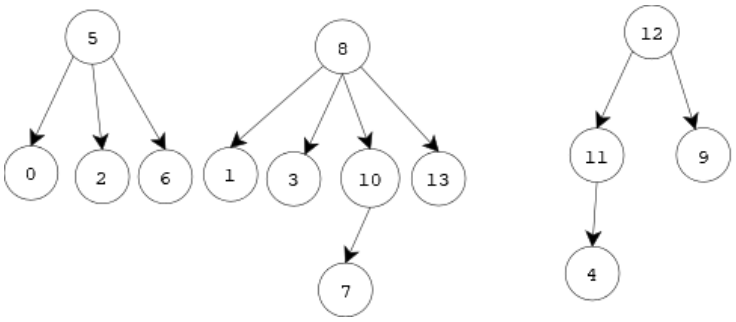


FIGURE 2 – before

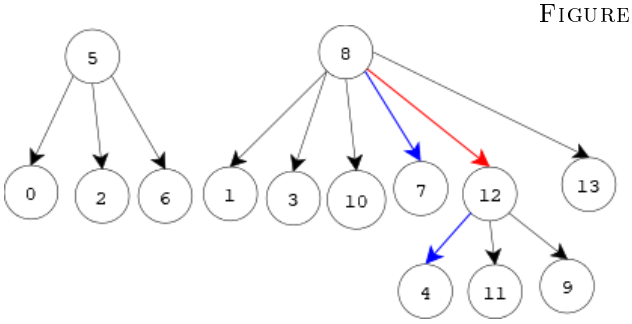


FIGURE 3 – after edge 7 – 4 added

***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 $0 < dmin \leq dmax$).

```
def dist_range(G, src, dmin, dmax):
    dist = [None] * G.order
    q = queue.Queue()
    q.enqueue(src)
    dist[src] = 0
    L = []
    while not q.isempty():
        x = q.dequeue()
        if dist[x] >= dmin:
            L.append(x)
        if dist[x] < dmax:
            for y in G.adjlists[x]:
                if dist[y] == None:
                    dist[y] = dist[x] + 1
                    q.enqueue(y)
    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

```
    def __get_cycle(G, x, parent):       # DFS on G from x, interrupted at first back
    edge found
        for y in G.adjlists[x]:          # parent: vertices marked with their parent
            if parent[y] == None:        # return first back edge found (x, y) or None
                get = __get_cycle(G, y, parent)
                if get != None:
                    return get
            else:
                if y != parent[x]:
                    return (x, y)
        return None

    def get_cycle(G):
        parent = [None] * G.order
        s = 0
        get = None
        while s < G.order and get == None:
            if parent[s] == None:
                parent[s] = -1
                get = __get_cycle(G, s, parent)
            s += 1
        L = []
        if get != None:
            (x, y) = get
            L = [x]
            while x != y:
                x = parent[x]
                L.append(x)
            L.append(L[0])
        return L
```

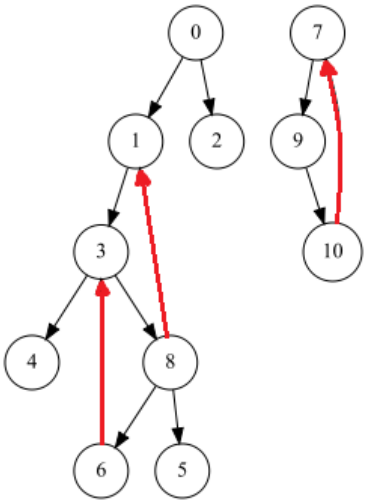
Version 2 : la fonction récursive construit le cycle à la remontée
De nombreuses manières de faire. La difficulté : ne plus ajouter de sommets lorsque le cycle est complet.

```
def __get_cycle2(G, x, M, p):
    """
       DFS on G from x
       M: mark vector (boolean)
       p: x's parent
       return (cycle, done):
       - cycle = the vertices of the first cycle found, [] if no cycle
       - done: boolean: is the cycle completed?
    """
    M[x] = True
    for y in G.adjlists[x]:
        if not M[y]:
            (cycle, done) = __get_cycle2(G, y, M, x)
            if cycle:
                if done:
                    return (cycle, True)
                if cycle[0] != y:
                    cycle.append(y)
                return (cycle, cycle[0] == y)
        else:
            if y != p:
                return ([y], False)
    return ([], False)

    def get_cycle_2(G):
        M = [False] * G.order
        for s in range(G.order):
            if not M[s]:
                cycle, done = __get_cycle2(G, s, M, -1)
                if cycle:
                    return cycle + [cycle[0]]
        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.