Algorithmique
Correction Partiel n° 3 (P3)

INFO-SPÉ - S3 – EPITA

19 décembre 2023 - 9 : 30

Solution 1 **(Warshall - Trouver-Réunir –** ***5 points*****)**

1. La matrice d'adjacence de G^* la **fermeture transitive** de G (1 = vrai, vide = faux) :

	0	1	2	3	4	5	6	7	8	9
0	×				1	1				
1		×	1				1	1		
2		1	×				1	1		
3				×					1	1
4	1				×	1				
5	1				1	×				
6		1	1				×	1		
7		1	1				1	×		
8				1					×	1
9				1					1	×

2. ✓ : Vecteur valide. Sinon les valeurs qui ne sont pas correctes sont surlignées.

P_1

0	1	2	3	4	5	6	7	8	9
5	6	9	8	5	-1	7	-1	-1	8

P_2

0	1	2	3	4	5	6	7	8	9
-1	2	-1	8	0	6	1	2	-1	3

P_3 ✓

0	1	2	3	4	5	6	7	8	9
4	-1	1	-1	-1	0	1	2	3	8

P_4

0	1	2	3	4	5	6	7	8	9
4	6	7	8	5	-1	-1	2	9	-1

OK si un seul des deux

3. Vecteur P après ajout de l'arête $\{1, 3\}$ (avec l'union pondérée et la compression des chemins) :

P

0	1	2	3	4	5	6	7	8	9
5	**7**	7	9	5	-3	7	**-7**	9	**7**

***Solution 2* (Longest Path – *10 points*)**

1. *Un peu d'aide*
   (a) *Forêt couvrante et arcs supplémentaires pour le parcours profondeur du graphe G_2 (figure 2) :* arc croisé / cross edge - arc avant / forward edge

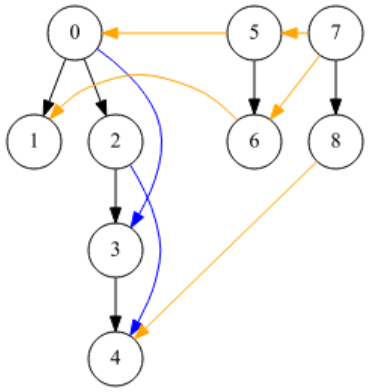

FIGURE 1 – DFS of G_2

   (b) Vecteur *long* :$\forall s \in S_2, long[s]$ = longueur du plus long chemin depuis s dans $G_2 =< S_2, A_2 >$.

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

2. **Spécifications :**
   La fonction `longest_path(G)`, avec G un graphe orienté, retourne :
   — si G n'est pas acyclique, la valeur `None`
   — si G est acyclique :
   ✓ la liste des sommets du plus long chemin dans G
   la longueur du plus long chemin dans G.

   One solution among many...

```python
def longest_path(G):
    Long = [None] * G.order
    Next = [None] * G.order
    lmax = -1
    for s in range(G.order):
        if Long[s] == None:
            if not __longest_path(G, s, Long, Next):
                return None
            if Long[s] > lmax:
                lmax = Long[s]
                x = s
    path = [x]
    for _ in range(lmax):
        x = Next[x]
        path.append(x)

    return path
```

Spécifications :

`__longest_path(G, x, Long, Next)`: DFS of `G` from `x`, returns a boolean = `G` is acyclic.
For any vertex `i` :

— `Long[i]` = `None` ⇒ `i` not marked, -1 ⇒ marked in prefix, otherwise the length of the longest path found from `i` (filled in suffix)

— `Next[i]` : the next vertex after `i` in the longest path

```python
def __longest_path(G, x, Long, Next):
    Long[x] = -1
    longest = 0
    for y in G.adjlists[x]:
        if Long[y] == None:
            if not __longest_path(G, y, Long, Next):
                return False
        else:
            if Long[y] == -1:
                return False
        if Long[y] + 1 > longest:
            longest = Long[y] + 1
            Next[x] = y
    Long[x] = longest
    return True
```

Solution 3 (Smallest Level – *5 points*)

Spécifications :

La fonction `smallest_level(`G`, src)` retourne la plus petite liste des sommets à une même distance (>0) de src dans le graphe non orienté connexe G.

Version avec vecteurs des distances

```python
def smallest(G, src):
    dist = [None] * G.order
    q = queue.Queue()
    q.enqueue(src)
    dist[src] = 0
    small = G.order
    d = 0
    L = []
    while not q.isempty():
        x = q.dequeue()
        if dist[x] > d: # new level
            if len(L) < small:
                small = len(L)
                Res = L
            d += 1
            L = []
        for y in G.adjlists[x]:
            if dist[y] == None:
                dist[y] = dist[x] + 1
                q.enqueue(y)
                L.append(y)
    return Res
```

Version avec deux files

```python
def smallest_2_queues(G, src):
    M = [False] * G.order
    q = queue.Queue()
    q.enqueue(src)
    M[src] = True
    q_next = queue.Queue()
    small = G.order
    L = []
    while not q.isempty():
        while not q.isempty():
            x = q.dequeue()
            for y in G.adjlists[x]:
                if not M[y]:
                    M[y] = True
                    q_next.enqueue(y)
                    L.append(y)
        # new level
        if L != [] and len(L) < small:
                    # L [] = last level
            small = len(L)
            Res = L
        L = []
        (q, q_next) = (q_next, q)
    return Res
```