Algorithmic
Correction Final Exam #3 (P3)

Undergraduate 2^{nd} year - S3 – Epita

December 17, 2022 - 9 : 30

Solution 1 **(Warshall - Union-Find –** ***4 points*****)**

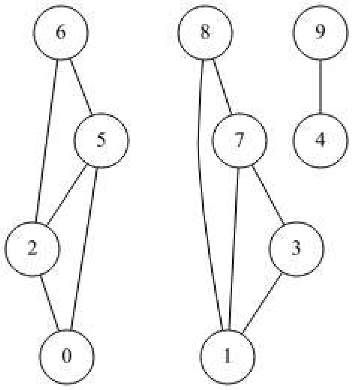

Figure 1: Graph G used

1. The adjacency matrix of G^* the **transitive closure** of G (no value = *false*, 1 = *true*):

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

2. ✓: Valid vectors. Otherwise the values that are not correct are highlighted.

P_1 ✓

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

P_2

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

P_3 ✓

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

P_4

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

3. Number of edges added to the new transitive closure G_2^* after adding $\{5, 1\}$ to G:

16 ($= 4 \times 4$)

Solution 2 **(Influencers –** ***5 points*****)**

Specifications:
The function `__eccentricity(`G`, `s`)` computes the eccentricity of s in G.

```python
def __excentricity(G, s, excMin):
    """
    return the minimum of
    - the excentricity of s in G
    - and excMin + 1
    """
    dist = [-1] * G.order
    q = queue.Queue()
    q.enqueue(s)
    dist[s] = 0
    while not q.isempty():
        x = q.dequeue()
        if dist[x] > excMin:
            return mini + 1
        for y in G.adjlists[x]:
            if dist[y] == -1:
                dist[y] = dist[x] + 1
                q.enqueue(y)
    return dist[x]
```

Another solution to stop the BFS (a little more optimized):

- stop the BFS at the first vertex whose distance is `excMin`
- this vertex and those in the queue (all at the same distance) must not have successors unmarked

Specifications:
The function `influencers(`G`)` returns the list of the *influencers* of the connected graph G.

```python
def influencers(G):
    excMin = __excentricity(G, 0)
    L = [0]
    for s in range(1, G.order):
        exc = __excentricity(G, s, excMin)
        if exc < excMin:
            L = [s]
            excMin = exc
        elif exc == excMin:
            L.append(s)
    return L
```

Solution 3 **(I want to be tree –** ***8 points*****)**

1. **Specifications:** The function `in_degrees(`G`)` returns the vector (a list in Python) of in-degrees of all vertices of the digraph G.

```python
def in_degrees(G):
    """
    in-degrees of vertices in G
    """
    Din = [0] * G.order
    for x in range(G.order):
        for y in G.adjlists[x]:
            Din[y] += 1
    return Din
```

2. Types of edges met during the depth-first search of the digraph G from r, when G is a tree rooted in r: Only tree (discovery) edges.

3. **Specifications:**

   The function `rooted_tree(G)` returns the vertex r if the digraph G is a tree rooted in r, the value `None` otherwise.

   First definition:

   - *a single root (in-degree == 0 and all other vertices reachable from it)*
   - *and no other edges than tree/discovery met during DFS from this root*

```python

def __istree(G, x, M):
    """
    dfs of G from x
    return -1 if G is not a tree, the number of vertices met otherwise
    """
    M[x] = True
    nb = 1
    for y in G.adjlists[x]:
        if not M[y]:
            n = __istree(G, y, M)
            if n == -1:
                return -1
            else:
                nb += n
        else:
            return -1
    return nb

def rooted_tree(G):
    """
    G: digraph
    if G is a rooted tree in r, return r, None otherwise
    """
    Din = in_degrees(G)
    r = None
    for s in range(G.order):
        if Din[r] == 0:
            if  r == None:
                r = s
            else:               # several roots
                return None
    if r == None:        # no root
        return None
    else:
        M = [False] * G.order
        if __istree(G, r, M) == G.order:
            return r
        else:
            return None
```

Definition 2:

- *only one root (in-degree == 0 and all other vertices reachable from it)*
- *all other vertices of in-degree = 1 (same as no other edges than tree ones during DFS)*

```python

def __nb_vertices(G, x, M):
    """
    dfs of G from x
    return number of vertices met
    """
    M[x] = True
    nb = 1
    for y in G.adjlists[x]:
        if not M[y]:     # useless here, as Din[y] == 1!
            nb  += __nb_vertices(G, y, M)
    return nb

def rooted_tree_2(G):
    Din = in_degrees(G)
    root = None
    for r in range(G.order):
        if Din[r] == 0:
            if root == None:
                root = r
            else:        # several roots
                return None
        elif Din[r] != 1:    # in-degree > 1
            return None
    M = [False] * G.order
    if __nb_vertices(G, root, M) != G.order:
        return None
    else:
        return root
```

Solution 4 (What does it do? – *3 points*)

1. `mystery`(G_{myst}):

   (a)

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

   (b) `what`(G_{myst}) returns: 7

2. `mystery`(G) returns -1 when G has a circuit.