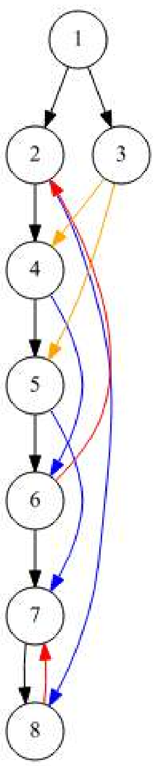
Algorithmique
Correction Partiel n^{o} 3 (P3)

Info-spé S3# – Epita

15 mai - 10 : 00

Solution 1 **(Forêt et ordres –** ***3 points*****)**

1. *Forêt couvrante :*



2. *Tableaux : ordres de rencontre des sommets en préfixe et suffixe* ***avec un compteur unique****.*

	1	2	3	4	5	6	7	8
pref	1	2	14	3	4	5	6	7
suff	16	13	15	12	11	10	9	8

Solution 2 **(Distances et centre –** ***6,5 points*****)**

1. **Spécifications :** La fonction `excentricity(`G`, `s`)` calcule l'excentricité de s dans G.

```python
def excentricity(G, src):
    dist = [-1] * G.order
    q = queue.Queue()
    q = queue.enqueue(src, q)
    dist[src] = 0
    while not queue.isEmpty(q):
        s = queue.dequeue(q)
        for adj in G.adjLists[s]:
            if dist[adj] == -1:
                dist[adj] = dist[s] + 1
                q = queue.enqueue(adj, q)
    return dist[s]
```

2. **Spécifications :** La fonction `center(G)` retourne le centre du graphe G.

```python
def center(G):
    excMin = excentricity(G, 0)
    c = 0
    for s in range(1, G.order):
        exc = distance(G, s)
        if exc < excMin:
            (L, excMin) = ([s], exc)
        else:
            L.append(s)
    return L
```

Solution 3 (I want to be tree – *5 points*)

Spécifications :

La fonction `isTree(G)` détermine si le graphe G est un arbre.

Version 1 : l'algo récursif marque les sommets avec le vecteur des pères. Il retourne un couple (nb sommets, booléen = sans cycle)

```python
def __isTree(G, s, P):
    nb = 1
    for adj in G.adjLists[s]:
        if P[adj] == None:
            P[adj] = s
            (n, tree) = __isTree(G, adj, P)
            nb += n
            if not tree:
                return (nb, False)
        else:
            if adj != P[s]:
                return (nb, False)
    return (nb, True)
#-----------------------------------------------------------------------------
def isTree(G):
    P = [None] * G.order
    P[0] = -1
    (nb, tree) = __isTree(G, 0, P)
    return tree and nb == G.order
```

Version2 : l'algo récursif marque les sommets avec un simple vecteur de booléens. Du coup le père du sommet courant est passé en plus en paramètre. L'algo ne retourne pas le nombre de sommets rencontrés : obligation de vérifier que tous sont marqués dans l'algo d'appel (moins optimal).

```python
def __isTree2(G, s, M, p):
    M[s] = True
    for adj in G.adjLists[s]:
        if not M[adj]:
            if not __isTree(G, adj, M, s):
                return False
        else:
            if adj != p:
                False
    return True
#-----------------------------------------------------------------------------
def isTree2(G):
    M = [False] * G.order
    if not __isTree2(G, 0, M, -1):
        return False
    for i in range(G.order):
        if not M[i]:
            return False
    return True
```

Solution 4 **(What is this ? – *5,5 points*)**

1. `build_graph(`G_4`, 5, 2, `NG`)` :

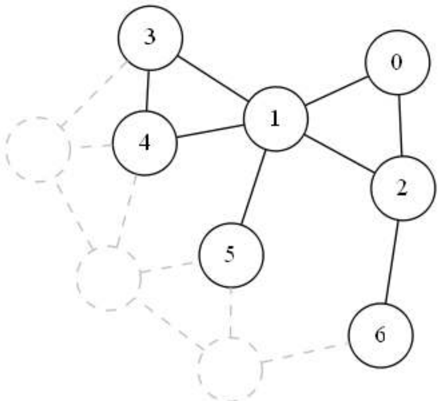
   (a) `dist`

1	2	3	4	5	6	7	8	9	10
-1	2	2	1	0	1	2	2	-1	-1

   (b) `map`

1	2	3	4	5	6	7	8	9	10
0	4	5	2	1	3	6	7	0	0

   (c) Le graphe résultat (NG) :



2. `build_graph(`G`, `s`, `n`, `NG`)` (G quelconque, $s \in G$, $n > 0$) :

   (a) `dist[i]` représente pour chaque sommet `i` atteignable en au plus n arêtes sa distance à la source (s)

   (b) `map` permet de renuméroter les sommets dans le nouveau graphe. *ou :* `map[i]` est le numéro du sommet i dans le nouveau graphe.

   (c) Le graphe NG est le sous-graphe obtenu à partir des sommets atteignables depuis s en au plus n arêtes.