Algorithmique
Partiel n° 3 (P3)

Info-spé - S3
Epita

17 décembre 2019 - 9 : 30

Consignes (à lire) :

- ☐ Vous devez répondre sur **les feuilles de réponses prévues à cet effet**.
  - — Aucune autre feuille ne sera ramassée (gardez vos brouillons pour vous).
  - — Répondez dans les espaces prévus, **les réponses en dehors ne seront pas corrigées** : utilisez des brouillons !
  - — Ne séparez pas les feuilles à moins de pouvoir les ré-agrafer pour les rendre.
  - — Aucune réponse au crayon de papier ne sera corrigée.
- ☐ La présentation est notée en moins, c'est à dire que vous êtes noté sur 20 et que les points de présentation (2 au maximum) sont retirés de cette note.
- ☐ **Le code :**
  - — Tout code doit être écrit dans le langage `Python` (pas de C, Caml, Algo ou autre).
  - — **Tout code `Python` non indenté ne sera pas corrigé.**
  - — Tout ce dont vous avez besoin (classes, fonctions, méthodes) est indiqué dans l'énoncé !
  - — Vous pouvez également écrire vos propres fonctions, dans ce cas elles doivent être documentées (on doit savoir ce qu'elles font).
    Dans tous les cas, la dernière fonction écrite doit être celle qui répond à la question.
- ☐ Durée : 2h00

Exercice 1 (Représentation et questions... – *2 points*)

Supposons un graphe simple G non orienté de 9 sommets représenté par la matrice d'adjacence suivante :

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

1. Donner deux propriétés des graphes que possède la fermeture transitive de G.
2. Donner les sommets dans l'ordre suffixe de rencontre pour un parcours en profondeur du graphe G. Comme de coutume, vous commencerez le parcours depuis le sommet 1 en considérant les successeurs en ordre numérique croissant.

Exercice 2 (Warshall - Trouver-Réunir – *4 points*)

Soit le graphe non orienté $G = < S, A >$, où les sommets sont numérotés de 0 à 13. L'algorithme Warshall vu en cours a permis de construire la matrice suivante (pas de valeur = *faux*, 1 = *vrai*) à partir de G :

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

1. Quelles sont les composantes connexes (ensembles de sommets) du graphe G ?
2. On applique les algorithmes `trouver` et `réunir` (versions non optimisées) à la liste des arêtes A de G. Parmi les vecteurs suivants, lesquels pourraient correspondre au résultat ?

P_1

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

P_2

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

P_3

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

P_4

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

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

Définition :
Un **arbre** est un graphe **connexe sans cycle**.

En utilisant **obligatoirement un parcours profondeur**, écrire la fonction `isTree` qui vérifie si un graphe non orienté est un arbre.

Exercice 4 (Distances et centre – *6 points*)

Définitions :

- La **distance** entre deux sommets d'un graphe est le nombre d'arêtes d'une **plus courte chaîne** entre ces deux sommets.
- On appelle **excentricité** d'un sommet x dans un graphe $G =< S, A >$ la quantité :

$$\text{exc}(x) = \max_{y \in S}\{distance(x, y)\}$$

- Le **rayon** d'un graphe est l'excentricité minimale de ses sommets, c'est-à-dire la plus petite distance à laquelle puisse se trouver un sommet de tous les autres.
- Le **centre** d'un graphe est formé de l'ensemble de ses sommets dont l'excentricité est le rayon du graphe (donc les sommets d'excentricité minimale).

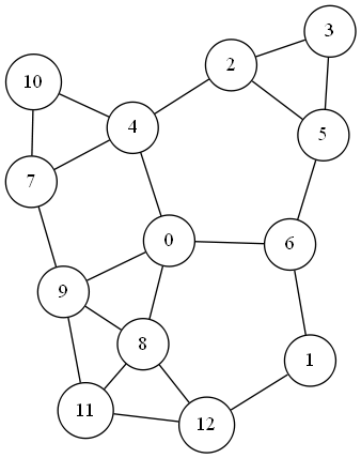
Écrire la fonction `center(G)` qui retourne le centre du graphe G (une liste).

Pour le graphe G_6 :

Les sommets 0, 4 et 6 ont pour excentricité 3.
Les sommets 3 et 11 ont pour excentricité 5.
Les sommets restants ont pour excentricité 4.

Le rayon de G_6 est donc 3 et son centre est constitué des sommets 0, 4 et 6.

```
>>> center(G6)
[0, 4, 6]
```



Graphe G_6

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

```python
import Graph, Queue

def __aux(G, v, m, res, count):
    q = Queue()
    q.enqueue(v)
    m[v] = True
    while not q.isempty():
        cur = q.dequeue()
        res[cur] = count
        count += 1
        for succ in G.adjlists[cur]:
            if not m[succ]:
                m[succ] = True
                q.enqueue(succ)
    return count

def mystery(G, src):
    m = [False] * G.order
    res = [0] * G.order
    count = __aux(G, src, m, res, 0)
    for v in range(G.order):
        if not m[v]:
            count = __aux(G, v, m, res, count)
    NG = Graph(G.order, directed=G.directed)
    for v in range(G.order):
        for s in G.adjlists[v]:
            if G.directed or res[v] <= res[s]:
                NG.addedge(res[v], res[s])
    return NG
```

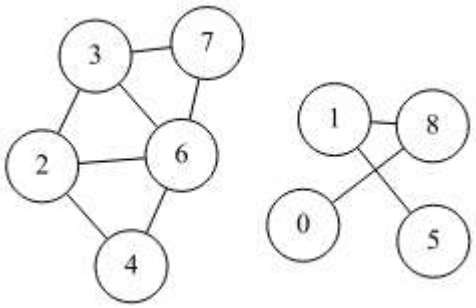


FIGURE 4 – Graphe G_4

1. Dessiner le graphe résultat de l'appel `mystery(`G_4`, 0)` (les sommets sont choisis en ordre croissant) avec G_4 le graphe de la figure 4.

2. Donner la liste des sommets dans l'ordre où ils ont été traités.

3. Soit un graphe non orienté possédant k composantes connexes. Combien de composantes contiendra le graphe obtenu par l'application de la fonction `mystery` ?

Annexes

Les classes `Graph` et `Queue` sont supposées importées.

Les graphes

Tous les exercices utilisent l'implémentation par listes d'adjacences des graphes.

Les graphes manipulés ne peuvent pas être vides.

```
class Graph:
    def __init__(self, order, directed = False):
        self.order = order
        self.directed = directed
        self.adjlists = []
        for i in range(order):
            self.adjlists.append([])
```

Les files

- `Queue()` returns a new queue
- q`.enqueue(`e`)` enqueues e in q
- q`.dequeue()` returns the first element of q, dequeued
- q`.isempty()` tests whether q is empty

Autres

- `range`
- sur les listes :
  - `len(L)`
  - `L.append()`
  - `L.pop()`
  - `L.insert(index, elt)`

Vos fonctions

Vous pouvez également écrire vos propres fonctions, dans ce cas elles doivent être **documentées** (on doit savoir ce qu'elles font).

Dans tous les cas, la dernière fonction écrite doit être celle qui répond à la question.