Algorithmique
Partiel n° 3 (P3)

Info-spé (S3)
Epita

18 décembre 2018 - 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 (Warshall - Union-Find – *3 points*)

Soit le graphe non orienté $G = < S, A >$, où les sommets sont numérotés de 0 à 9. 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
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

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
2	-1	-1	1	1	2	2	1	1	4

P_2

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

P_3

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

P_4

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

Exercice 2 (Dans les profondeurs de la forêt couvrante – *2 points*)

Représenter (dessiner) la forêt couvrante associée au parcours en profondeur du graphe `G`. *Ajouter aussi les autres arcs en les qualifiant à l'aide d'une légende explicite.* On considérera le sommet 1 comme base du parcours, les sommets devant être choisis en ordre numérique croissant.

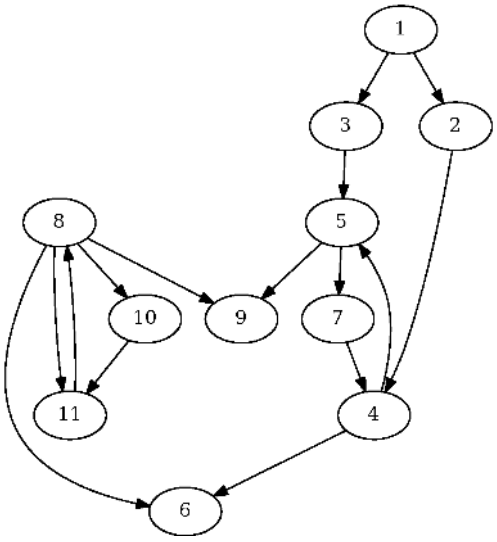

FIGURE 1 – Un graphe orienté

Exercice 3 (Composantes – *3 points*)

Écrire la fonction `components` qui détermine les composantes connexes d'un graphe non orienté.

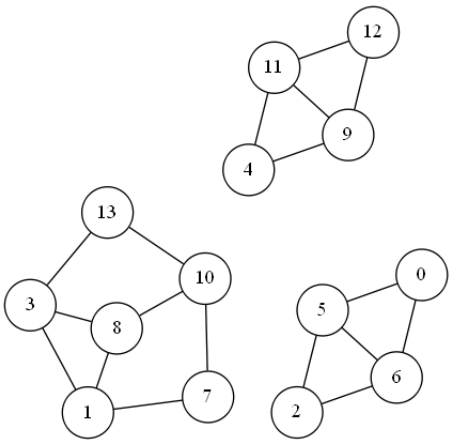

FIGURE 2 – Graphe `G1`

La fonction retourne un couple :
- Le nombre de composantes connexes du graphe ;
- le vecteur des composantes : un vecteur qui indique pour chaque sommet le numéro de la composante à laquelle il appartient.

Exemple d'application, avec `G1` le graphe de la figure 2 :

```
>>> components(G1)
(3, [1, 2, 1, 2, 3, 1, 1, 2, 2, 3, 2, 3, 3, 2])
```

Exercice 4 (Diamètre – 5 *points*)

Définitions :
- La **distance** entre deux sommets d'un graphe est le nombre d'arêtes d'une **plus courte chaîne** entre ces deux sommets.
- Le **diamètre** d'un graphe est la **plus grande distance** qu'il puisse exister entre deux de ses sommets.

Lorsque le graphe est un arbre, le calcul du diamètre est simple :
- À partir d'un sommet quelconque s_0, on cherche un sommet s_1 de distance maximale à s_0.
- À partir de s_1, on cherche un sommet s_2 de distance maximale à s_1.
- La distance entre s_1 et s_2 est le diamètre du graphe.

Écrire la fonction `diameter` qui calcule le diamètre d'un graphe qui est un arbre.

Exercice 5 (Euler – 6 *points*)

Un graphe non orienté est dit *Eulérien* si on peut trouver une *chaîne eulérienne* passant par toutes ses arêtes une fois et une seule ou *un cycle eulérien* (dont les extrémités coïncident).

Le théorème d'Euler énonce qu'un graphe non orienté admet une *chaîne eulérienne* si et seulement si il est **connexe** et admet **zéro ou deux sommets de degrés impairs**. Si tous les sommets sont de degrés pairs, il s'agit d'un *cycle eulérien*.

Écrire une fonction, utilisant **obligatoirement un parcours profondeur** qui détermine si un graphe non orienté **simple** est *eulérien*, c'est-à-dire possède une chaîne ou un cycle eulérien.

Un peu d'aide : comment connaître le degré d'un sommet avec l'implémentation par listes d'adjacence ?

Exercice 6 (What is this ? – *3 points*)

```
1  def __what(G, x, d):
2      lc = 0
3      for y in G.adjlists[x]:
4          if d[y] == -1:
5              d[y] = d[x] + 1
6              lc = max(lc, __what(G, y, d))
7          else:
8              if d[y] + 1 < d[x]:
9                  lc = max(lc, d[x] - d[y] + 1)
10     return lc
11
12 def what(G):
13     d = [-1] * G.order
14     lc = 0
15     for x in range(G.order):
16         if d[x] == -1:
17             d[x] = 0
18             lc = max(lc, __what(G, x, d))
19     return (lc, d)
```

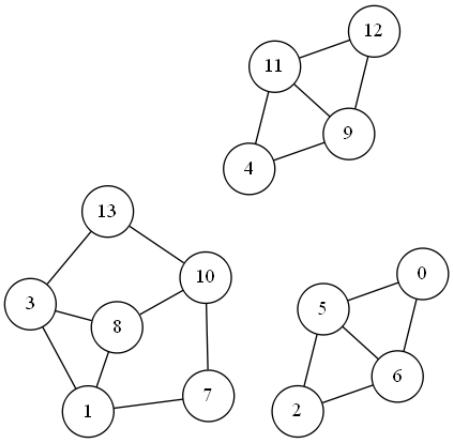

FIGURE 4 – Graphe G_4

1. Quel sera le résultat retourné par `what`(G_4) avec G_4 le graphe ci-dessus, dont les listes de successeurs sont en ordre croissant de numéros ?

2. Que représente le vecteur `d` ?

3. **Bonus :** Que représente la valeur `lc` ?

Annexes

Les classes `Graph` et `Queue` sont supposées importées. Les graphes manipulés ne peuvent pas être vides.

Les graphes

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

```
class Graph:
    def __init__(self, order, directed = False):
        self.order = order
        self.directed = directed
        self.adjlists = []
        for i in range(order):
            self.adjlists.append([])
    def addedge(self, src, dst):
        self.adjlists[src].append(dst)
        if not self.directed and dst != src:
            self.adjlists[dst].append(src)
```

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

Les fonctions que vous pouvez utiliser :

- sur les listes : `len`
- `range`.

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.

Rappel :

Les entiers ne peuvent pas être "passés par référence" en Python...