Algorithmique
Partiel n° 3 (P3)

INFO-SPÉ - S3
EPITA

17 décembre 2021 - 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 (Dans les profondeurs de la forêt couvrante – *3 points*)

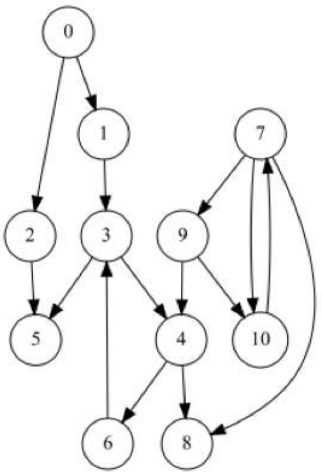

FIGURE 1 Un graphe orienté

1. Représenter (dessiner) la forêt couvrante associée au parcours profondeur du graphe de la figure 1. *Ajouter aussi les autres arcs en les qualifiant à l'aide d'une légende explicite.* On considérera le sommet 0 comme base du parcours, les sommets devant être choisis en ordre numérique croissant.

2. Remplir les vecteurs d'ordres de rencontre en préfixe et suffixe, établis avec un compteur unique commençant à 1, correspondants au parcours de la question précédente.

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

Soit le graphe non orienté $G = < S, A >$, où les sommets sont numérotés de 0 à 10. 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
0	1						1	1			
1		1	1	1						1	
2		1	1	1						1	
3		1	1	1						1	
4					1	1			1		1
5					1	1			1		1
6	1						1	1			
7	1						1	1			
8					1	1			1		1
9		1	1	1						1	
10					1	1			1		1

1. On applique les algorithmes `trouver` et `réunir` (versions optimisées) à la liste des arêtes A de G.
Parmi les vecteurs suivants, lesquels pourraient correspondre au résultat ? Lorsque le vecteur est faux, entourer les valeurs qui posent problème.

P_1

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

P_2

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

P_3

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

P_4

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

2. On ajoute l'arête $\{1, 0\}$ au graphe G. Combien d'arêtes seront ajoutées à la fermeture transitive de G ?

Exercice 3 (Coloring – *8 points*)

En théorie des graphes, *colorer un graphe* signifie attribuer une couleur à chacun de ses sommets de manière à ce que deux sommets adjacents soient de couleurs différentes. Le nombre minimal de couleurs pour colorer un graphe est appelé *nombre chromatique* $\chi(G)$.

Une autre manière de voir : Soit le graphe non orienté $G = < S, A >$ si $\chi(G) = k$ alors S peut être partitionné en k ensembles $S_i, i \in [1, k]$ tels que pour toute arête $(u, v) \in A \Rightarrow u \in S_i$ et $v \in S_j$, avec $i \neq j$. Aucune arête ne doit relier deux sommets d'un même ensemble.

1. Quel est le *nombre chromatique* (le nombre minimum de couleurs, $\chi(G_i)$) des graphes suivants ?
   Les couleurs seront représentées par des entiers. Pour chaque graphe, remplir le tableau des couleurs : des entiers dans $[1, \chi(G_i)]$.

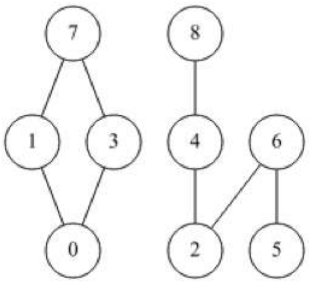

FIGURE 2 G_1

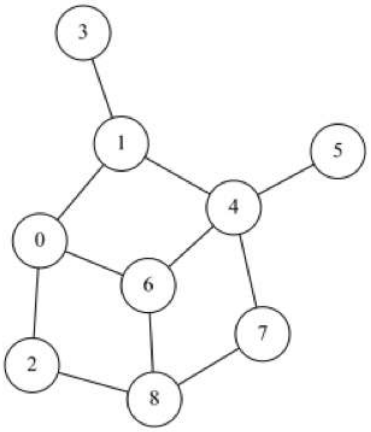

FIGURE 3 G_2

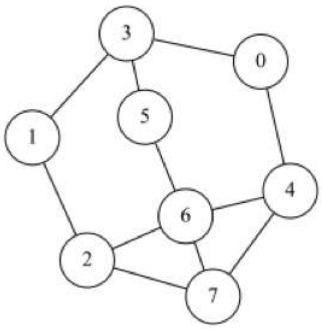

FIGURE 4 – G_3

2. Écrire la fonction `two_coloring(G)`, qui utilise **obligatoirement** un parcours profondeur, qui vérifie si $\chi(G) = 2$ (le graphe peut être coloré avec 2 couleurs).

Exercice 4 (Fake News – *5 points*)

Le graphe G ci-dessous représente le réseau social "we're all friends" : 2 sommets reliés sont "amis" sur ce réseau.

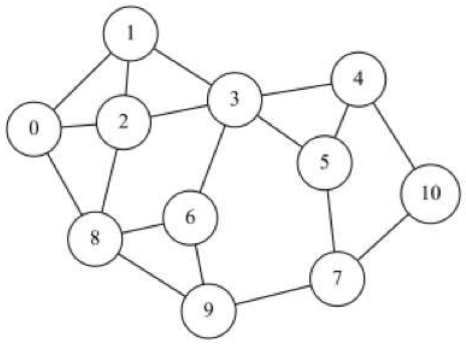

FIGURE 5 – G : "we're all friends"

Une personne (la source) de ce réseau envoie un message à tous ses amis qui le lisent immédiatement. Dès qu'une personne lit le message pour la première fois, elle l'envoie à son tour à tous ses amis exactement 3 minutes plus tard.

L'information diffusée a au départ un "indice de fiabilité" donné. À chaque fois que l'information est diffusée, elle perd 1 point de fiabilité. Si une personne reçoit l'information de plusieurs amis, elle ne tiendra compte que de la première.

Par exemple, dans notre réseau "we're all friends", 0 dispose d'une information d'indice de fiabilité 6.
1, 2 et 8 recevrons l'information avec un indice 5 ;
3, 6 et 9 recevrons quant à eux l'information avec un indice 4.
— ...
Une information sera considérée comme "fake news" si elle perd 50% ou plus de son indice de fiabilité.

Écrire la fonction `fakenews(`G`, `*src*`, `*truth*`)` qui retourne la liste des sommets de G, graphe non orienté **connexe**, qui ont reçu une "fake news" à partir d'une information d'indice de fiabilité $truth > 2$ diffusée par *src*.

Exemples d'applications avec le graphe G de la figure 5 (l'ordre des sommets dans le résultat n'est pas important) :

```
>>> fakenews(G, 0, 6)
[4, 5, 7, 10]

>>> fakenews(G, 0, 4)
[3, 4, 5, 6, 7, 9, 10]

>>> fakenews(G, 10, 7)
[0]

>>> fakenews(G, 6, 7)
[]
```

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. Il n'y a pas de liaisons multiples ni boucles.

```
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`
- `min, max`

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

Et n'importe quel opérateur...

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.