Algorithmique
Contrôle n° 4 (C4)

Info-spé (S4)
Epita

5 mars 2019 - 14 : 45

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 (Cut points, cut edges – *5 points*)

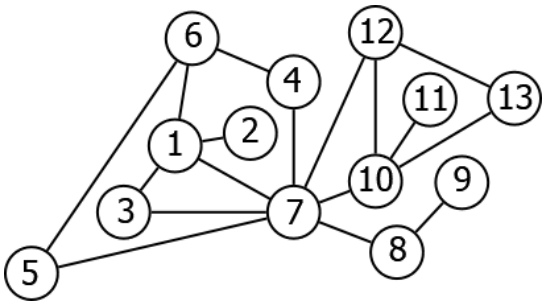


FIGURE 1 – Graphe G_1

1. Quels sont les points d'articulation de G_1 ?
2. Quels sont les isthmes (ponts) de G_1 ?
3. Donner, sous la forme de listes d'arêtes, les composantes biconnexes de G_1 (une composante par ligne).
4. Donner le tableau des valeurs *prefixe* et *plushaut*, de chaque sommet du graphe G_1, obtenu lors du parcours profondeur main gauche du graphe G_1. Vous utiliserez pour *plushaut* la formule suivante :

$$plushaut(x) = min \begin{cases} prefixe(x) & \\ plushaut(y) & \forall\ (x,y) \text{ arc couvrant} \\ prefixe(z) & \forall\ (x,z) \text{ arc retour} \end{cases}$$

   Comme d'habitude, la racine du parcours est le sommet 1 et les successeurs sont traités en ordre croissant.

Exercice 2 (I want to be a tree – *8 points*)

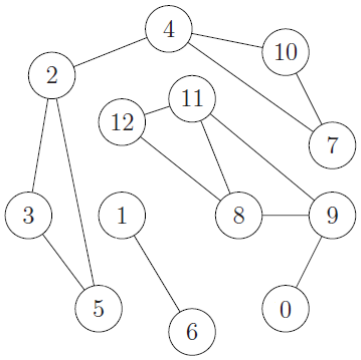


FIGURE 2 – Not a tree yet

Le but de cet exercice est de transformer un graphe quelconque en arbre en le modifiant le moins possible.

1. Donner deux définitions d'un graphe qui est un *arbre* ?

2. On effectue un **parcours profondeur** d'un graphe non orienté :
   (a) Quelles sont les arêtes qui peuvent être enlevées **lors du parcours** du graphe sans augmenter le nombre de composantes connexes ?
   (b) Appliquer au graphe de la figure 2 : donner la liste des arêtes que l'on supprimera lors du parcours profondeur en choisissant les sommets dans l'ordre croissant (y compris pour les successeurs).

3. Pendant le parcours profondeur, on attribue à chaque sommet un numéro de composante connexe (de 1 à k, s'il y a k composantes) :
   (a) Combien d'arêtes faut-il ajouter pour rendre le graphe connexe ?
   (b) Comment, **lors du parcours**, savoir quelles arêtes ajouter pour rendre le graphe connexe ?
   (c) Appliquer au graphe de la figure 2 : donner la liste des arêtes à ajouter pour le rendre connexe.

4. Écrire la fonction qui
   — construit le vecteur des composantes connexes *cc* du **graphe initial** ;
   — ajoute au graphe les arêtes qui permettent de le rendre connexe ;
   — supprime du graphe les arêtes "inutiles" (sans augmenter le nombre de composantes).

Exercice 3 (Graphe réduit – *4 points*)

Soit G un graphe orienté admettant k composantes fortement connexes : $C_0, C_1, \ldots, C_{k-1}$. On définit le *graphe réduit* de G (noté G_R) par $G_R = < S_R, A_R >$ avec :

— $S_R = \{C_0, C_1, \cdots, C_{k-1}\}$

— $C_i \rightarrow C_j \in A_R \Leftrightarrow$ Il existe au moins un arc dans G ayant son extrémité initiale dans la composante fortement connexe C_i et son extrémité terminale dans la composante fortement connexe C_j.

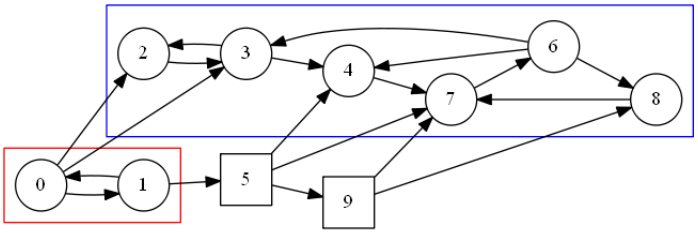


FIGURE 3 – Graphe G_1

On désire construire le graphe réduit d'un graphe G. On dispose de la liste des composantes fortement connexes de G (chaque composante est une liste des sommets qui la constitue).

Par exemple pour le graphe de la figure 3, la liste des composantes est :

```
scc = [[2, 3, 4, 6, 7, 8], [9], [5], [0, 1]]
```

Écrire la fonction `condensation(`G`,` *scc*`)` qui à partir du graphe G et *scc* sa liste de composantes fortement connexes retourne le graphe réduit G_R ainsi que le vecteur des composantes : un vecteur qui pour chaque sommet de G indique à quelle composante il appartient (le numéro du sommet dans G_R).

Par exemple avec le graphe G_1 de la figure 3 :

```
>>> (Gr, comp) = condensation(G1, scc)
>>> comp
[3, 3, 0, 0, 0, 2, 0, 0, 0, 1]
```

Exercice 4 (Graphes et mystère – *3 points*)

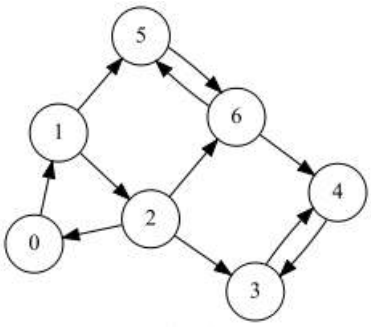


FIGURE 4 – Graphe G_2

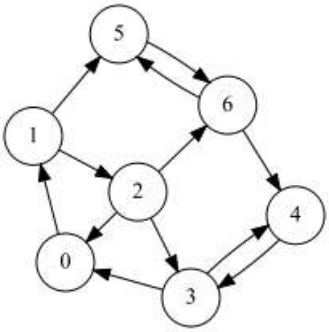


FIGURE 5 – Graphe G_3

```
1        def __test(G, x, p, c):
2            c += 1
3            p[x] = c
4            rx = p[x]
5            for y in G.adjlists[x]:
6                if p[y] == 0:
7                    (ry, c) = __test(G, y, p, c)
8                    if ry == -1:
9                        return (-1, c)
10                   rx = min(rx, ry)
11               else:
12                   rx = min(rx, p[y])
13
14           if rx == p[x]:
15               if p[x] != 1:
16                   return (-1, c)
17
18           return (rx, c)
19
20       def test(G):
21           p = [0] * G.order
22           c = 0
23           (r, c) = __test(G, 0, p, c)
24           return (r != -1) and (c == G.order)
```

On suppose que dans le graphe passé en paramètre les listes d'adjacence sont triées en ordre croissant.

1. Pour chacun des appels suivants, avec G_2 et G_3 les graphes des figures 4 et 5 :
   — combien d'appels à `__test` ont été effectués ?
   — quel est le résultat retourné par `test` ?

   (a) `test(`G_2`)`

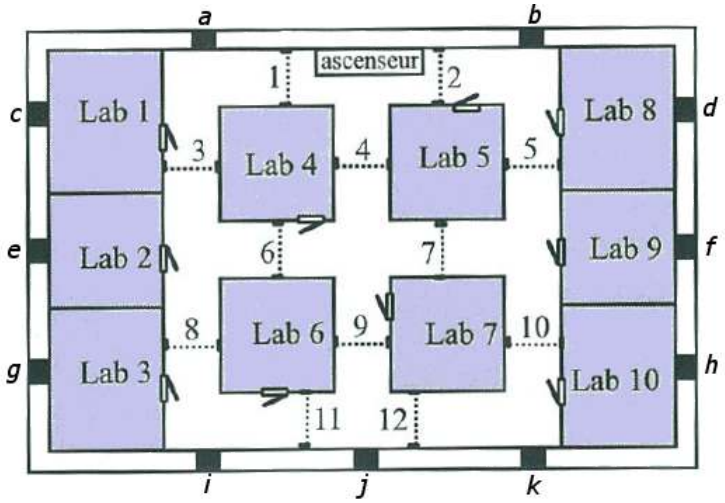
   (b) `test(`G_3`)`

2. Soit G un graphe orienté. Quelle information est retournée par `test(`G`)` ?

Exercice 5 (Il faut sauver Algernon – *Bonus*)

Algernon, la célèbre souris, a été kidnappée par un membre du centre de recherche, qui l'a ramenée à son laboratoire. Mais Algernon est maline et a réussi à s'échapper, profitant de l'ouverture de la porte du laboratoire. Il faut retrouver Algernon : pendant la nuit, elle a pu aller n'importe où et maintenant, elle peut être dans un laboratoire, coincée dans une bouche d'aération.

Voici le plan du centre de recherche :
— Les 10 laboratoires (Lab 1 à Lab 10), un par chercheur ;
— les bouches d'aération, représentées par les carrés noirs (de a à k) : une souris peut y entrer, mais pas en sortir ; de plus chaque bouche d'aération mène à une impasse ;
— l'ascenseur, qui mène directement dehors ;
— en pointillé, les détecteurs de mouvements (numérotés de 1 à 12).



Les détecteurs de mouvements sont un peu archaïques : ce sont de simples indicateurs mécaniques qui comptent le nombre de passages (on ne peut donc pas savoir à quel moment le mouvement a été détecté).

Voici le nombre de détections effectuées pendant la nuit :

n° détecteur	1	2	3	4	5	6	7	8	9	10	11	12
nb détections	2	1	3	3	1	3	0	2	2	3	2	1

Quel chercheur est l'auteur du rapt ? Où se trouve Algernon maintenant ?

1. (a) Quel chercheur est l'auteur du rapt ?
   (b) Où se trouve Algernon maintenant ?
2. Justifier (utiliser le plan donné pour illustrer la justification).

Annexes

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

Les graphes

```python
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)

    def removeedge(self, src, dst):
        self.adjlists[src].remove(dst)
        if not self.directed and dst != src:
            self.adjlists[dst].remove(src)
```

Les files

- `Queue()` retourne une nouvelle file
- q`.enqueue(`e`)` enfile s dans q
- q`.dequeue()` retourne le premier élément de q, défilé
- q`.isempty()` teste si q est vide

Autres

- 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.