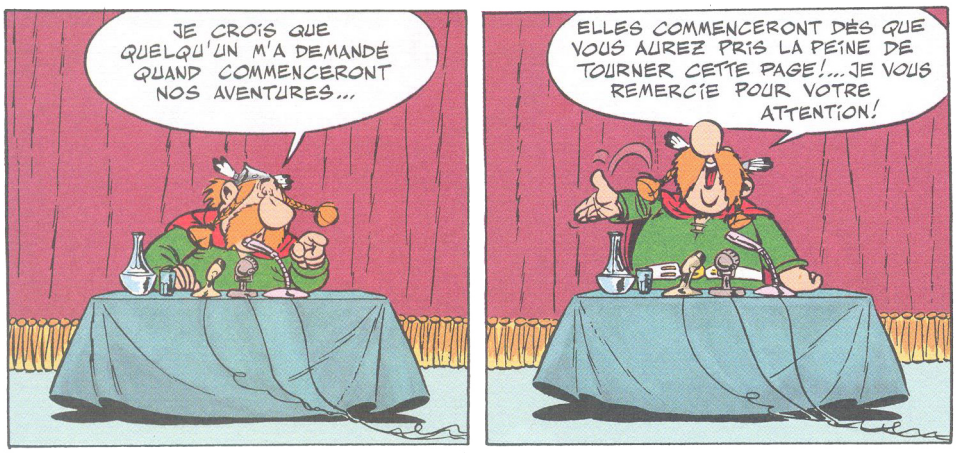
Algorithmique
Contrôle nº 4 (C4)

Info-spé (S4)
Epita

6 mars 2018 - 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 – *3 points*)

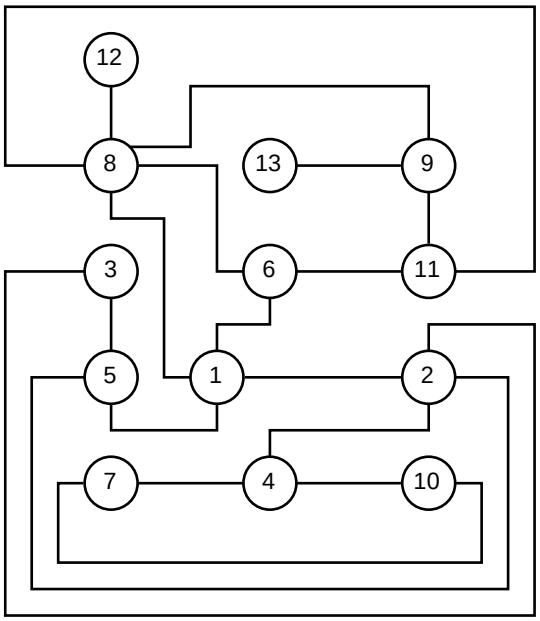


FIGURE 1 – Graphe G_1

1. Donner la forêt couvrante associée au parcours en profondeur du graphe G_1 à partir du sommet 1, en choisissant les sommets dans l'ordre croissant. Vous devez y ajouter tous les arcs du graphes, avec une légende explicite indiquant la nature de chaque arc.
2. Quels sont les points d'articulation de G_1 ?
3. Quels sont les isthmes (ponts) de G_1 ?

Exercice 2 (CFC et graphe réduit – *5 points*)

Soit G un graphe orienté admettant k composantes fortement connexes : C_1, C_2, ..., C_k. On définit le *graphe réduit* de G (noté G_R) par $G_R = < S_R, A_R >$ avec :
- $S_R = \{C_1, C_2, \cdots, C_k\}$
- $C_i \to C_j \in A_R \Leftrightarrow$ 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.

1. **Graphe → graphe réduit**

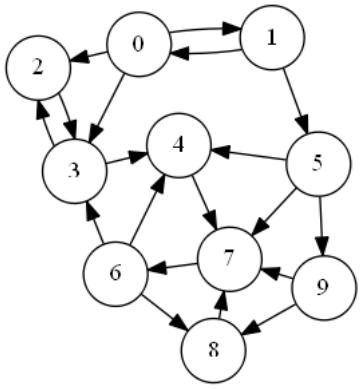


FIGURE 2 – Graphe G_2

(a) Donner, en les numérotant, les composantes fortement connexes du graphe G_2 (figure 2).

(b) Représenter le graphe réduit du graphe G_2 (utiliser les numéros de la question a).

(c) L'ajout d'un seul arc peut-il rendre le graphe G_2 fortement connexe ? Justifier.

2. **Graphe réduit → graphe**
   Soit le graphe G_3 dont on ne connaît que les composantes fortement connexes (données dans le tableau *comp* ci-dessous), et le graphe réduit (figure 3).

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
comp	1	1	1	1	2	2	2	2	3	3	3	3	4	4	4	4	5	5	5	5	6	6

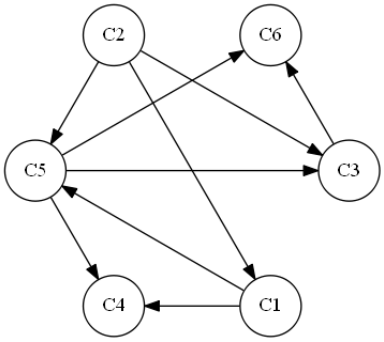


FIGURE 3 – Graphe réduit de G_3

(a) Quels sont les sommets qui ne sont pas atteignables depuis le sommet 0 ?

(b) Parmi les chemins suivants, quels sont ceux qui ne peuvent pas exister dans G_3 ?
- $3 \rightsquigarrow 7$
- $4 \rightsquigarrow 21$
- $18 \rightsquigarrow 2$
- $11 \rightsquigarrow 15$

(c) Donner le nombre minimum d'arcs qu'il faut ajouter pour rendre le graphe G_3 fortement connexe.

Exercice 3 (Indicateurs globaux de connexité – *6 points*)

Les indicateurs globaux de connexité mesurent le degré de fragmentation d'un graphe en composantes connexes séparées les unes des autres. Soient N le nombre de sommets d'un graphe, k son nombre de composantes connexes et $n_1, \cdots, n_k$ le nombre de sommets de chacune des composantes connexes ($n_1 + n_2 + ... + n_k = N$). On peut définir deux indices de connexité dont les valeurs sont comprises entre 0 (graphe vide) et 1 (graphe connexe).

- Indice de connexité simple : $IC_1 = (N - k)/(N - 1)$
- Indice de connexité pondéré : $IC_2 = (n_1^2 + n_2^2 + ... + n_k^2)/(N^2)$

Par exemple sur le graphe de la figure 4 :

- $IC_1 = (14 - 3)/(14 - 1) = 0,9166..$
- $IC_2 = (4^2 + 4^2 + 6^2)/14^2 = 0,3469..$

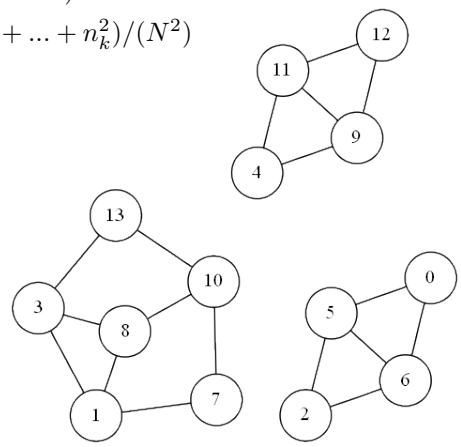


FIGURE 4 – Graphe à 3 composantes

1. En terme de probabilités, quelle information sur le graphe nous donne l'indice de connexité pondéré ?

2. En utilisant **un parcours profondeur**, écrire les fonctions permettant de calculer les deux indices de connexités (simple et pondéré) d'un graphe (représenté par listes d'adjacences).

Exercice 4 (Fortement connexe ? – *7 points*)

Le but de l'exercice est d'écrire une fonction vérifiant la forte connexité d'un graphe. Pour cela nous allons utiliser l'algorithme de Tarjan simplifié.

Rappel principe Tarjan :

On utilise un parcours profondeur, dans lequel on numérote les sommets en ordre préfixe de rencontre.

Pour chaque sommet x on calcule la valeur $retour(x)$, qui associe au sommet x l'ordre préfixe d'un sommet de la composante fortement connexe de x, rencontré avant x, s'il en existe un, ou bien x lui-même. La valeur de *retour* est définie de la manière suivante :

$$retour(x) = min\{pref[x], retour(y), pref[z]\}$$

où le minimum est calculé sur

- tous les sommets y descendants de x dans la forêt couvrante,
- tous les sommets z tels que $x \to z$ est un arc retour, ou un arc croisé si la racine de la composante fortement connexe de z est un ancêtre de x.

Une fois le parcours à partir de x terminé, on vérifie si la valeur de retour du sommet x est toujours identique à sa valeur d'ordre préfixe. Si c'est le cas, le sommet x est alors une *racine de composante*.

Questions :

1. Si le graphe est fortement connexe, quelle(s) propriété(s) doit avoir la première *racine de composante* trouvée lors du parcours ?
2. Écrire la fonction `is_strong(`G`)` qui vérifie si le graphe orienté G est fortement connexe.

Annexes

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

Les 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()` 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.