Algorithmique
Partiel n° 3 (P3)

Info-spé S3#
Epita

15 mai - 10 : 00

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 en **annexe** !
  - — 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 (Forêt et ordres – *3 points*)

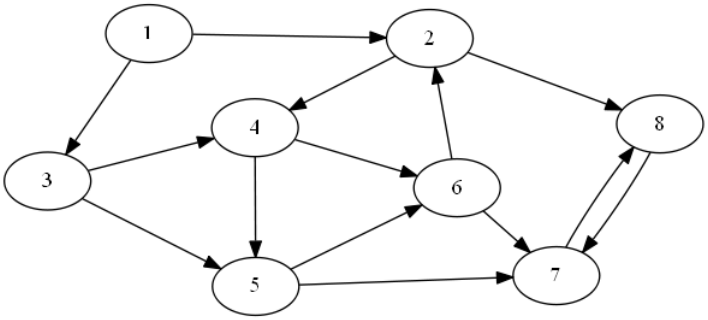


FIGURE 1 – Graphe pour parcours profondeur

Appliquer le parcours profondeur depuis le sommet 1 sur le graphe de la figure 1 (les sommets seront choisis dans l'ordre croissant).

1. Construire la forêt couvrante correspondante. Ajouter à la forêt obtenue les différents types d'arcs rencontrés lors du parcours, en les qualifiant d'une légende explicite.
2. Soit $pref$ et $suff$ les tableaux indiquant les ordres de rencontre préfixes et suffixes des sommets (établis avec un compteur unique commençant à 1) lors du parcours profondeur. Remplir les tableaux d'ordre préfixe $pref$ et suffixe $suff$ obtenus.

Exercice 2 (Distances et centre – *6,5 points*)

Définitions :

- La **distance** entre deux sommets d'un graphe est le nombre d'arêtes d'une **plus courte chaîne** entre ces deux sommets.
- 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).

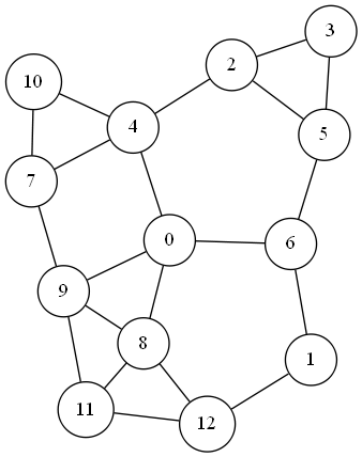
Écrire la fonction `center(G)` qui retourne le centre du graphe G (une liste).

Pour le graphe G_1 :

Les sommets 0, 4 et 6 ont pour excentricité 3.
Les sommets 3 et 10 ont pour excentricité 5.
Les sommets restants ont pour excentricité 4.

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

```
>>> center(G1)
[0, 4, 6]
```



Graphe G_1

Exercice 3 (I want to be tree – *6 points*)

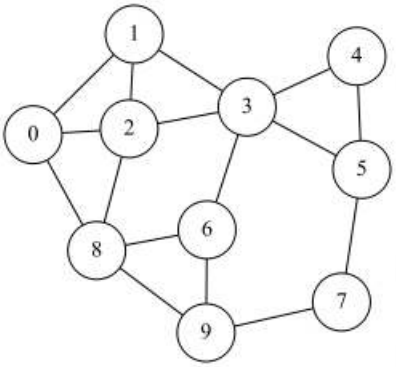
Définition :
Un **arbre** est un graphe **connexe sans cycle**.

1. Lors du parcours profondeur d'un graphe non orienté :
   (a) Comment vérifier que le graphe est sans cycle ?
   (b) Comment vérifier que le graphe est connexe ?
2. Écrire la fonction `isTree` qui vérifie si un graphe non orienté est un arbre.

Exercice 4 (What is this ? – *4,5 points*)

Soit la fonction suivante :

```
1  def buildGraph(G, s, n):
2      map = [None] * G.order
3      dist = [-1] * G.order
4      NG = graph.Graph(1)
5      dist[s] = 0
6      map[s] = 0
7      q = queue.Queue()
8      q.enqueue(s)
9      while not q.isempty():
10         s = q.dequeue()
11         for adj in G.adjlists[s]:
12             if (dist[adj] == -1) and (dist[s] < n):
13                 dist[adj] = dist[s] + 1
14                 map[adj] = NG.order
15                 NG.addvertex()
16                 q.enqueue(adj)
17             if dist[adj] != -1:
18                 NG.addedge(map[s], map[adj])
19     return (NG, dist, map)
```



Graphe G_2

1. On appelle cette fonction avec `build_graph(`G_2`, 4, 2)` (avec G_2 le graphe ci-dessus, dont les listes de successeurs sont en ordre croissant de numéros).
   (a) Remplir le vecteur `dist`.
   (b) Remplir le vecteur `map`.
   (c) Dessiner le graphe résultat (NG).
2. `build_graph(`G`, `s`, `n`)` est appelée avec G un graphe quelconque (non vide), s un sommet de G, et n un entier naturel non nul.
   (a) Que représente le vecteur `dist` ?
   (b) À quoi sert le vecteur `map` ?
   (c) Que représente le graphe NG ?

Annexes

Les classes `Graph` et `Queue` sont supposées importées. Les graphes manipulés 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 addvertex(self):
        self.order += 1
        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

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.