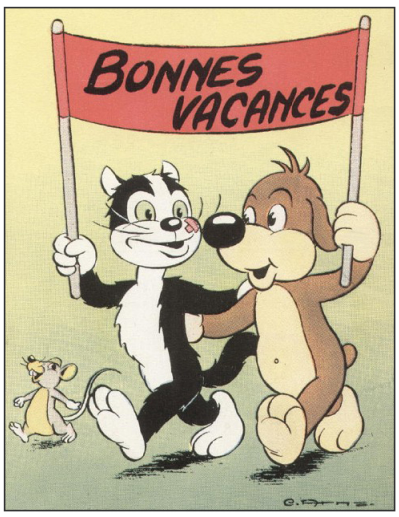
Algorithmique
Partiel n° 3 (P3)

Info-spé (S3#)
Epita

5 juin 2017 - 9h

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



Des graphes

Exercice 1 (Graphes : Sans circuit... – *3 points*)

1. Quelle est, au niveau de la classification de ses arcs, la particularité d'un graphe sans circuit ?

2. Soit $G =< S, A >$ un graphe sans circuit, soient les tableaux *os* et *op* contenant, respectivement, les numéros d'ordre suffixe et préfixe de tous les sommets du graphe G obtenus lors du parcours en profondeur de G. Démontrer que pour une paire quelconque de sommets distincts $u, v \in S$, s'il existe un arc dans G de u à v, alors $os[v] < os[u]$.

Exercice 2 (Dans les profondeurs de la forêt couvrante – *4 points*)

On se propose d'effectuer un parcours profondeur sur un graphe orienté. L'objectif est de construire la forêt couvrante du parcours, d'ajouter les différents types d'arcs rencontrés et de calculer l'ordre suffixe de rencontre de chaque sommet.

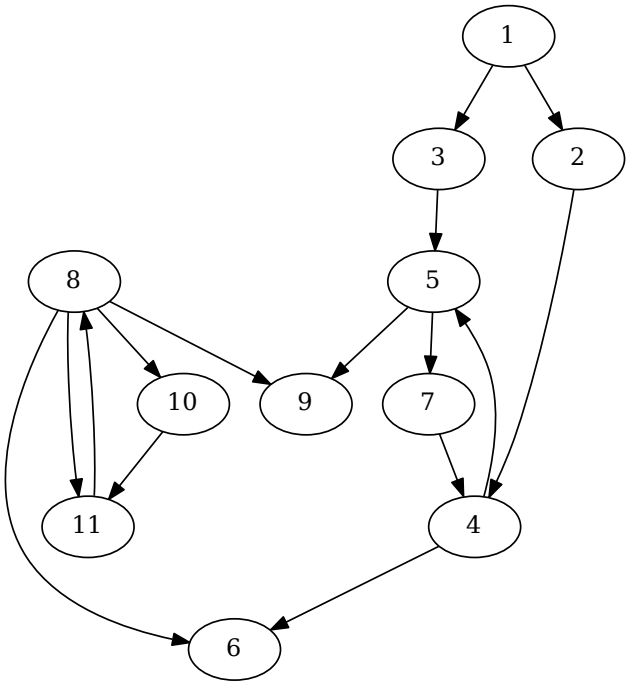


FIGURE 1 – Un graphe orienté

1. Construire la forêt couvrante du graphe de la figure 1 correspondant à un parcours profondeur depuis le sommet numéro 1. On rencontrera les sommets dans l'ordre croissant. Ajouter à la forêt obtenue les différents types d'arcs rencontrés lors du parcours.

2. Remplir les vecteurs de pères et d'ordres suffixes pour chacun des sommets, correspondants au parcours de la question précédente. Les racines de la forêt seront indiquées par la valeur -1 dans le vecteur de pères.

Exercice 3 (Composantes – *4 points*)

Écrire la fonction `components` qui détermine les composantes connexes d'un graphe non orienté en utilisant obligatoirement un parcours profondeur.

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.

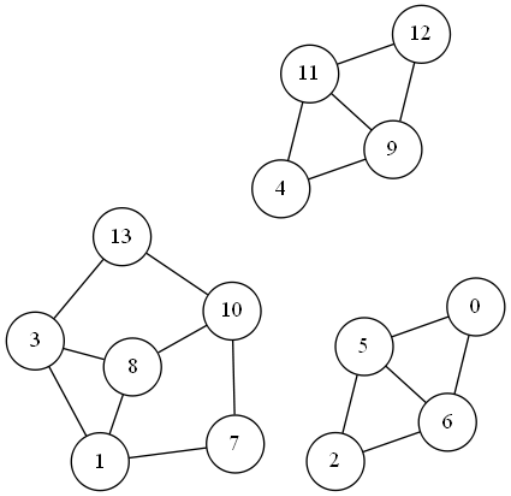


FIGURE 2 – Graphe `G1`

Example 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 (Chemin – *6 points*)

1. Quel parcours utiliser pour trouver un chemin (ou une chaîne) le plus court (en nombre de liaisons) entre deux sommets donnés dans un graphe ?

2. On cherche un chemin (le plus court possible) entre deux sommets dans un graphe orienté. Écrire la fonction `path(`G`, `*src*`, `*dst*`)` qui recherche un chemin dans G entre les sommets *src* et *dst* : si un chemin a été trouvé, la liste des sommets le composant devra être retournée (dans l'ordre, voir l'exemple ci-dessous), une liste vide sinon.

Example d'application, avec `G2` le graphe de la figure 3 :

```
>>> path(G2, 0, 8)
[0, 1, 5, 9, 8]
```

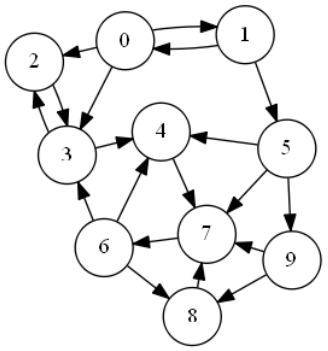


FIGURE 3 – Graphe `G2`

Des arbres

Exercice 5 (Nœuds rouges – *3 points*)

Soit A un arbre 2-3-4, écrire la fonction `nbRed(`A`)` qui retourne le nombre de nœuds rouges que contiendrait un arbre bicolore équivalent à A.

Annexes

Les classes, fonctions et méthodes peuvent être utilisées sans préfixe.

Les graphes

Les graphes manipulés sont non vides !

```
class Graph:
    def __init__(self, order, directed = False):
        self.order = order
        self.directed = directed
        self.adjLists = []
        for i in range(order):
            self.adjLists.append([])
```

Arbres 2-3-4 (B-Arbres de degré 2)

```
    class BTree:
        degree = 2

        def __init__(self, keys=None, children=None):
            self.keys = keys if keys else []
            self.children = children if children else []

        @property
        def nbKeys(self):
            return len(self.keys)
```

Les fonctions que vous pouvez utiliser :

Les files

- `Queue()` returns a new queue
- `enqueue(`e`, `q`)`
- `dequeue(`q`)`
- `isEmpty`q`)`

Autres

- sur les listes :
  - `L.append(x)` : ajoute `x` en fin de `L`
  - `L.insert(i, x)` : ajoute `x` en position `i` dans `L`
- `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.