TP no 01

Les buts de ce premier TP sont les suivants :
- reprendre à programmer en **python**,
- de façon que votre chargé de TP puisse évaluer votre niveaux avec ce langage,
- utiliser **python** comme outil pour l'enseignement.

Un peu de programmation

Récupérez, depuis votre solution du TP 4 du cours Informatique 1, votre code **python** des algorithmes de parcours en largeur et de parcours en profondeur. Éventuellement, implémentez à nouveau ces algorithmes, en utilisant le code vu en cours et disponivle sur le site du cours.

Exercice 1. Modifiez votre code afin d'avoir une seule implantation de l'algorithme. Cette implantation sera paramétrée de façon à s'exécuter en tant que parcours en largeur ou en profondeur, selon une variable booléenne que l'on passera comme argument optionnel de la procédure.

Exercice 2. Ajoutez à votre implantation des algorithmes un paramètre optionnel `root`.
Si `root` n'est pas `None`, alors l'algorithme explorera seulement la partie du graphe atteignable depuis le sommet `root` passé en paramétré. Autrement, l'algorithme explorera le graphe en entier.

Exercice 3. Réutilisez le code écrit jusqu'à maintenant afin de :
- Définir une fonction qui prend en input un graphe et deux sommets, et retourne la distance entre ces deux sommets,
- Définir une fonction qui teste si un graphe passé en paramètre est acyclique. NB : si un graphe ne contient pas de cycles, alors est un foret. Le foret recouvrent retournée est donc le graphe lui même. Donc il faudra vérifier à chaque fois que l'ensemble de voisins d'un sommet en cours de traitement sont tous blancs sauf éventuellement un. De que cette condition n'est pas vérifiée, on s'arrête et on retourne `False`.
- Définir une fonction qui teste si un graphe passé en paramètre est biparti. NB : un graphe est biparti ssi il ne contient pas un cycle de longueur impair. Il faudra donc modifier la fonction précédente afin de détecter les cycles de longueur impair.

Visualisation de graphes

L'environnement **graphviz** permet d'obtenir de visualisation de graphes sans se soucier de la disposition exacte des noeuds. Le langage **dot** est le langage de représentation de graphes utilisé par **graphviz**.

Exercice 4. Écrivez une fonction qui prend en paramètre un graphe (décrit par listes d'adjacences) et produit la représentation du graphe dans le langage **dot**. Par exemple, pour un graphe dont la représentation par liste d'adjacences est

```
{0: [1, 2],
 1: [2, 0],
 2: [0, 1],
 3: [4, 6],
 4: [5, 3],
 5: [6, 4],
 6: [3, 5]}
```

la représentation dans le langage **dot** est la suivante :

```
graph {
  0 -- 1
  0 -- 2
  1 -- 2
  3 -- 4
  3 -- 6
  4 -- 5
  5 -- 6
}
```

Exercice 5. Modifiez la fonction écrite dans l'exercice précèdent, de façon que l'on puisse aussi ajouter les couleurs des sommets (blanc pour non découvert, gris pour en cours de traitement, noir pour traité) pendant le parcours du graphe, et les arête de l'arbre recouvrent. Par exemple, à la quatrième itération de la boucle sur le graphe précèdent, on pourra avoir le résultat suivant :

```
graph {
  0 [style=filled, fontcolor=white, fillcolor = black]
  1 [style=filled, fontcolor=white, fillcolor = black]
  2 [style=filled, fontcolor=white, fillcolor = black]
  3 [style=filled, fillcolor = gray]

  0 -- 1
  0 -- 2
  1 -- 2
  3 -- 4
  3 -- 6
  4 -- 5
  5 -- 6

  1 -- 0 [constraint=false,color= red]
  2 -- 0 [constraint=false,color= red]
}
```

Exercice 6. Écrivez une fonction qui prend en paramètre le graphe et le sauve sous forme **dot** dans un fichier.
Passez cette fonction en paramètre à la procédure de parcours de graphe, de façon à engendrer des fichiers dans le langage **dot** représentant, étape par étape, l'état du graphe étant exploré.

Graphes orientés

Exercice 7. Écrivez une version récursive de l'algorithme de parcours en profondeur qui manient des dictionnaires de temps de découverte et temps de fin de l'exploration d'un noeud.
Utilisez ce code pour définir une fonction qui détecte si un graphe orienté contient un circuit (cycle orienté).

Exercice 8. Écrivez une fonction qui prends en paramètre un graphe orienté, et, si ce graphe ne contient pas de circuits, alors retourne un tri topologique du graphe.

Exercice 9. Utilisez les exercices précédents pour définir une fonction qui calcule les composantes fortement connexes d'un graphe orienté passé en paramètre.