TD 2

Graphes (2) : Programmation Python

Nous fournissons une archive avec du code "à trous". Pour afficher les graphes, il y a une dépendance au logiciel `graphviz` *et son interface python 3*

2.0.1 Graphes non orientés

EXERCICE #1 ▸ **Décortiquons la classe fournie**
Regarder l'implémentation de la classe `Graph` dans le fichier `LibGraphes.py`. Cette classe fournit un constructeur, et des méthodes de parcours (non implémentées), une méthode heuristique de coloriage, et une méthode d'affichage utilisant Graphviz.

1. Dans `Main.py`, modifier le graphe fourni et vérifier que l'affichage correspond. *Il est normal que les fonctions de parcours ne fonctionnent pas, elles ne sont pas implémentées*
2. Discuter de l'intérêt d'utiliser une classe pour les Graphes.

EXERCICE #2 ▸ **Parcours**
Implémenter les deux parcours : en largeur (BFS) et en profondeur (DFS) *à partir d'un noeud distingué* en remplissant les trous "`TODO`" dans le code fourni de la bibliothèque. [1].

EXERCICE #3 ▸ **Composantes connexes**
Implémenter la recherche des composantes connexes.

EXERCICE #4 ▸ **Passage d'une représentation à une autre**
Les fonction utilitaires de changement de représentation sont explicitement au programme de Terminale, les questions de conception de cet exercice un peu moins . . .

Discuter des choix d'implémentation (critiquer !), et proposer des modifications de la librairie pour disposer des deux implémentations :

1. Quel type interne pour les matrices d'adjacence ?
2. Comment garder les deux représentations à jour ? Est-ce utile ? qu'est-ce que cela impose sur le code des fonctions déjà implémentées ?
3. Implémenter les méthodes de transformation d'une représentation à une autre.

2.1 Graphes Orientés

EXERCICE #5 ▸ **Une nouvelle classe ?**
Discuter des choix d'implémentation pour réaliser "à partir" de la classe fournie une librairie de graphes orientés. Implémenter une des solutions discutées.

EXERCICE #6 ▸ **Composantes fortement connexes**
Dans des graphes orientés on appelle composante fortement connexe un sous-graphe maximal pour la propriété "pour tout couple (u, v) de noeuds dans ce sous-graphe, il existe un chemin (orienté) de u à v."

EXERCICE #7 ▸ **Facultatif**
Implémenter l'algorithme de clôture transitive avec les produits de matrice. Tester.

EXERCICE #8 ▸ **Digicode (Graphe Eulérien) - facultatif**
Doc : `https://fr.wikipedia.org/wiki/Graphe_eul%C3%A9rien`

Implémenter le problème de recherche de séquence minimale pour ouvrir un digicode à 4 chiffres :

1. ce code à trou est automatiquement généré à partir du code prof solution avec le magnifique script `generate-skeletons` que vous pouvez trouver là :`https://gitlab.com/moy/generate-skeletons`

— Montrer que le problème se ramène à un problème de recherche de circuit eulérien.
— Implémenter les fonctions suivantes :
  1. `condEuler(self)` qui teste les conditions d'Euler.
  2. `findEulerianCircuit()` qui retourne (si il existe) un circuit eulérien.
— Répondre à la question. On écrira aussi une fonction qui teste qu'une séquence de chiffres donnée contient bien tous les codes.

2.1.1 Graphes Pondérés

Les graphes pondérés ne sont pas explicitement au programme de Terminale, mais certains algorithmes de flot sont des bons exemples de programmation dynamique.

EXERCICE #9 ▸ **Plus court chemin avec Dijsktra - facultatif**
On vous fournit une classe graphe pondéré (et `Main2.py`, implémentez l'algo de Dijkstra (cf slides).

— Implementez `mini(Q,d)` qui retourne un élément de Q qui minimise d. (d dictionnaire, Q ensemble)
— Les initialisations de l'algo sont déjà faites, il vous reste à implémenter la boucle.
— Améliorez l'affichage.
— Vérifiez l'algo sur plusieurs exemples "faits à la main".

EXERCICE #10 ▸ **Floyd-Warshall - facultatif**
Implémenter les plus courts-chemins avec Floyd Warshall. Implémenter la variante pour calculer la clôture transitive d'un graphe orienté (cf slides).