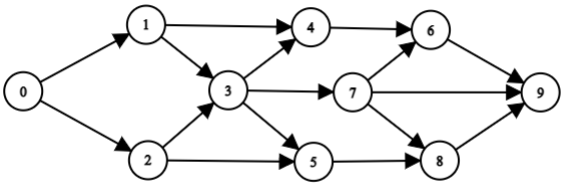
TD/TP Structure de données - Graphes

M. Tellene

1 Appréhender la structure de données graphe

Exercice 1
Sur feuille, représenter le graphe suivant sous forme de dictionnaire d'adjacence



Exercice 2
Donner la liste des sommets parcourus pour l'algorithme du parcours en profondeur sur le sommet 3.

Donner la liste des sommets parcourus pour l'algorithme du parcours en largeur sur le sommet 1.

Exercice 3
Représenter la version non orienté du graphe précédent sous forme de dictionnaire d'adjacence. Donner la liste des sommets parcourus pour l'algorithme du parcours en profondeur sur le sommet 3 sur ce nouveau graphe.

2 Manipuler d'un graphe

Organisation : aller dans votre dossier personnel, puis dans votre dossier « structure de données » (ou « SD »), dans ce dossier, créer un dossier « graphe », c'est ce dossier qui contiendra le travail fait lors de ce TP.

Durant ce TP, vous créerez une librairie contenant des méthodes applicables aux graphes. Cette librairie sera à rendre et sera notée. Vous pouvez vous aider mais un travail personnel est attendu.

On rappelle qu'un graphe est une structure de données composée de sommet de lien. Ces liens peuvent être des arcs, dans le cas d'un graphe orienté, ou bien des arêtes, dans le cas d'un graphe non orienté.

Vous récupérerez le fichier `LibGrapheEleve.py`. Ce fichier contient une classe `Graphe`. Vous devrez compléter et rendre fonctionnelle la classe `Graphe` avec les éléments indiqués.

Le fichier `fichier_test.py` est, comme son nom l'indique, un fichier de test pour la classe `Graphe`. Afin d'utiliser ce fichier, il vous suffit de récupérer le code et lancer le programme.

Voici, à titre indicatif, le barème de notation :

Méthode	Points
`__init__`	1
`verification_integrite`	3
`liste_sommets`	1
`liste_aretes`	3
`voisin`	1
`ajouter_sommet`	1.5
`ajouter_arete`	2
`suppression_sommet`	1.5
`suppression_arete`	1
`parcours_profondeur_iteratif`	4
`est_atteignable_de`	1
`parcours_largeur_iteratif`	4
`composant_connexe`	4
Total	27

3 Des méthodes supplémentaires

Voici deux méthodes pour aller plus loin avec cette classe `Graphe`. Il n'est pas obligatoire de les implémenter, mais le faire peut vous donner un bonus de points.

— `presence_cycle(self, depart)` : méthode renvoyant `True` si un cycle, ayant pour point de départ le sommet `depart`, existe, `False` sinon

— `coloration(self, K)` : méthode qui renvoie une coloration, si celle-ci est possible, du graphe avec `K` couleurs différentes. Dans le cas d'un graphe non colorable, la méthode devra renvoyer `False`

Afin de simplifier la compréhension et l'écriture d'un tel algorithme, vous baserez votre code sur la proposition de Kempe. Des explications sur le problème de coloration de graphe et un exemple de l'algorithme sont donnés. La méthode devra renvoyer un dictionnaire dans lequel la clé sera un sommet et la valeur sera la couleur associée. La couleur sera symboliser par un nombre.