IQ02E020 Exploration algorithmique d'un problème

SAE 2.02

Thème en 2024 : Graphes Orientés Valués

Contact : mikal.ziane@u-paris.fr ou mikal.ziane@lip6.fr ou mikal.ziane@gmail.com

Attention : le non-respect des dates limites indiquées ci-dessous entrainera des points de pénalités sur la note.

Introduction

Dans le cadre de la SAE 2.02 vous devez réaliser en Java une application permettant de manipuler des graphes comme indiqué dans une série de tests unitaires qui vous seront. Une application possible est la détermination du plus court chemin (en minutes) pour aller d'une station du métro parisien à une autre.

Vous vous répartirez en équipes de 4 étudiants (plus ou moins 1 si nécessaire) du même groupe de TP (201 par exemple) de façon à ce qu'il y ait 3 équipes par groupe : par exemple 4+4+3 = 11 ou 5+4+4 = 13.

Un projet privé github devra être créé pour l'équipe et votre chargé de TP devra être invité comme collaborateur. Un README.txt devra être défini avec la liste des membres de l'équipe et une synthèse de ce qui a été fait. Il n'y aura **pas** de rapport par ailleurs. Le chargé de TP devra avoir accès à tout le projet et notamment aux *commits*. Le Readme indiquant les membres de l'équipe et l'invitation du chargé de TD doivent être faites pour mercredi 3 avril matin 9h.

La note de ce travail ne sera pas individualisée, si l'équipe fonctionne sans problème majeur, mais lors des DST un exercice porteront sur ce travail et la note (individuelle) aura le même coefficient.

Ce travail est composé d'une série d'exercices qui portent sur la notion de sous-typage et de polymorphisme qui sont vus dans l'enseignement Développement à Objets. Tout le développement sera fait en Java.

Une recette/soutenance aura lieu lors de la dernière séance de TP : le projet devra donc être rendu sur github jusqu'au **lundi 27 mai matin 8h**. La partie I devra être déposée sur github pour le lundi 22 avril 8h. Il n'y aura aucun report.

Pour chacune des deux parties de la SAE, du code vous sera fourni qui inclura des tests unitaires qui devront être exécutés avec succès pour toutes les classes de graphe mentionnées plus haut. Vous ne devez **pas** changer le code qui vous a été fourni ni renommer quoi que ce soit pour faciliter la correction.

Consultez régulièrement Moodle et discord pour les dernières informations sur la SAE.

Partie I Représentation des graphes orientés valués

La notion de sous-typage permet de masquer les différentes façons d'implémenter un graphe (matrice d'adjacence, liste d'adjacence etc.) tout en permettant de manipuler des graphes malgré tout.

Les différentes façons d'implémenter un graphe seront simplement codées chacune dans une classe dédiée qui implémentera deux interfaces **`IGrapheConst`,** qui sera utilisée par Dijkstra, et IGraphe qui sera utilisée plus largement notamment pour construire les graphes.

Les nœuds seront identifiés par des chaines de caractères. Les graphes ne contiendront pas de valuation négative de façon à permettre l'application de Dijkstra.

Liste d'arcs

Écrivez une classe **`GrapheLArcs`** qui représente un graphe à l'aide d'une liste d'arcs. On doit savoir le nombre de nœuds du graphe donc tous les nœuds sont déjà connus : ce sont les entiers de 1 à n. A vous de décider comment représenter un arc.

Matrice d'adjacence

Écrivez une classe **`GrapheMAdj`** qui représente un graphe à l'aide d'une matrice d'adjacence.

Listes d'adjacence

Écrivez une classe **`GrapheLAdj`** qui représente un graphe par des listes d'adjacence.

Partie II Algorithme du plus court chemin de Dijkstra et étude comparative

Implémentez l'algorithme du plus court chemin de Dijkstra sur l'interface `IGraphe` **sans utiliser explicitement** les classes de graphes.
Il est conseillé d'utiliser un Fibonacci heap, mais ceci sera expliqué ultérieurement.
Comparez ensuite l'efficacité en temps et en espace de votre implémentation de Dijkstra pour les classes de graphes de la première partie. Chaque représentation sera évaluée pour des graphes avec un nombre plus ou moins grand de nœuds et d'arcs.
Vous pouvez ajouter votre propre représentation du graphe (une 4^{e} classe donc) si vous pensez que cela peut améliorer l'efficacité de votre algorithme, mais ce dernier devra tout de même rester générique c'est-à-dire n'utiliser explicitement que l'interface IGraphe.
Des graphes vous seront fournis grâce à un programme de génération aléatoire de Denis Poitrenaud à partir de la bibliothèque https://graphstream-project.org/
Des exemples de graphes aléatoires sont visibles là : https://graphstream-project.org/doc/Generators/