TD9- Composition vs Inheritance: Application to a Case Study on Graphs

Sujet de TD basé sur *Graphs and Graph Search* par Karthikeyan Bhargavan et Philippe Chassignet, école Polytechnique.
N'hésitez pas à regarder le sujet original qui étudie davantage l'algorithmique et permet des visualisations, mais ne vous perdez pas.
Pour des raisons de temps, nous nous limiterons à utiliser les graphes dans un cadre applicatif et à réfléchir aux structures de données adéquates en fonction du problème adressé.

Objectif applicatif : Il s'agit de reconstruire les objets correspondants aux villes stockées dans un fichier txt et ensuite de reconstruire la carte de France et de déterminer les chemins entre les villes.

Code donné : Les codes suivants ont été donnés par ce sujet de TD initial.
Ils ont à peine été modifiés, pour vous simplifier la tâche, mais il reste plusieurs petits soucis de langue, etc. Attention cependant, en particulier, le code de *CityParser* a été repris pour éliminer plusieurs problèmes à l'usage, méfiez-vous si vous choisissez de partir de la version originale… vous aurez du travail en plus. Une classe *CityParserTest* a été ajoutée…

Objectif pédagogique : Dans ce TD, vous êtes complétement libre sur les choix de structures données et même d'algorithmique, mais bien sûr vous devez utiliser des structures de graphes, soit par héritage, soit par composition.
Il est possible que vous ayez à les redéfinir, la forme originale des graphes vus en TD ne stockant dans les nœuds que des string et non des objets. Vous avez d'autres choix possibles, à vous de voir.
En tant que tel, ce TD n'est pas à rendre. Nous pensons néanmoins qu'il présente de nombreux intérêts pédagogiques dont celui, **de vous donner confiance en votre capacité à coder** une solution à un problème pas si facile que cela.

Remarques :

Vous allez travailler avec de grands graphes.

- Il est possible que **certaines fonctions récursives** ne passent pas à l'échelle. Il faut alors que vous révisiez vos codes pour passer les méthodes problématiques en itératif, par exemple, en créant une pile dans laquelle vous mettez les éléments à traiter.
- Pour d'autres fonctions, vous aurez des problèmes de performance. Quand vous utilisez certaines structures comme des HashMap, initialisez la taille initiale pour ne pas avoir en plus des temps de redimensionnements.

1. Construire un graphe de villes

Vous commencez par construire la carte sous la forme d'un graphe dont les villes sont reliées par des arcs que si la distance entre les villes est inférieure à 12000m (12km). C'est une approximation, on considère qu'il existe une route directe entre deux villes distantes de moins de 12km, vous pourrez bien sûr faire varier ce seuil en fonction de vos tests.
Le fichier ***fr.txt*** vous est donné, il contient les descriptions des villes : nom, longitude, latitude.
La classe *City* contient, elle, une fonction de calcul de distance.

Remarque : Il y a énormément de villes qui ont le même nom, et en plus, le fichier de départ contient les différentes versions des données associées aux villes. Nous vous conseillons de commencer par réduire la liste des villes obtenues par **public void** readAll(Collection<City> cities). Vous pouvez bien sûr faire d'autres choix.

Par exemple :

```
CityParser cp = new CityParser(path + "fr.txt");
List<City> cities = new ArrayList<>();
cp.readAll(cities);

List<City> afterRemoving =
CityParser.removeDuplicates(cities);
```

A la fin vous devez obtenir un graphe de villes.
A vous de tester vos codes…

J'ai personnellement obtenu : 58064 nœuds (villes) et 1168198 Arcs (routes supposées entre les villes de moins de 12km).

2. Calculez le nombre de groupes de villes qui sont à plus de 12 km

Puis vous calculez le nombre de groupes de villes qui ne sont pas connectées à d'autres, donc dont la distance à d'autres villes est supérieure à 12 km.
Par exemple :

- La majorité des villes sont connectées. J'ai personnellement un groupe de 57070 villes connectées
- Les villes de Corse forment un autre groupe de 940 villes
- Les villes de l' Île d'Yeu un groupe de 6 villes
- Autour de Lacanau il semble y avoir un groupe de 14 villes
- A la pointe Bretonne etc.

J'ai personnellement obtenu : 12 groupes.

3. Calculez le chemin le plus court entre deux villes

Faîtes varier votre calcul de la longueur minimale pour connecter deux villes afin de comprendre les résultats « étonnants ». Il est aussi possible que certaines longitudes/latitudes soient un peu fausses (j'en ai identifié et corrigé mais il en reste c'est certain).

Nice et Paris étant connectées, de même que Nice et Marseille vous pouvez calculer le chemin, après avoir bien su regarder le chemin le plus court à vols d'oiseaux entre Nice et Antibes par exemple.

4. Pour aller plus loin (l'implémentation n'est pas demandée)

Vous envisagez de découper la carte de France entre vos commerciaux… Que faîtes-vous ?
Vous voulez les aider et vous leur proposez un chemin qui passe par toutes les villes sans jamais repasser deux fois par la même ville. Que faîtes-vous ?