Programmation Orientée Objet
IG2I, LE2 – 2018-2019

TP7 – Routage

Diego Cattaruzza, Philippe Kubiak, Maxime Ogier

Contexte et objectif du TP

Ah le nord ! Partie trois ! On disait qu'au nord ils boivent beaucoup de bière. Bon, toutes les caisses sont encore et toujours au dépôt... Elles attendent vos indications pour être livrées.

Dans ce TP, nous allons optimiser le planning de livraison pour faire faire des économies à Monsieur Houblon. Si nous lui proposons un planning moins cher, le gain se répercutera sur le prix de la bière même, et tout le monde sera gagnant ! Allons-y alors : mettons-nous au boulot !

Ce TP permet d'illustrer les notions suivantes :

- utilisation des collections en Java ;
- des notions d'algorithmique.

On s'attachera en particulier dans ce TP à commenter proprement le code et à garantir que le code fonctionne et respecte les spécifications données.

Bonnes pratiques à adopter

Pour réaliser ce TP, il vous est très fortement recommandé d'utiliser la Javadoc pour comprendre le fonctionnement des méthodes que vous appelez. Par ailleurs, il est aussi important que votre propre code soit commenté au format Javadoc (commentaire commençant par `"/**"` au-dessus des définitions des attributs et méthodes). Ceci vous permet (1) de spécifier rigoureusement le comportement d'une méthode au moment où vous l'implémentez, (2) de savoir ce que fait une méthode sans avoir besoin de regarder son code (qui d'ailleurs peut ne pas être accessible), et donc (3) de faciliter le partage et la réutilisabilité du code.

Créez vos classes en mettant toujours les attributs en premier, suivis par les constructeurs (avec le constructeur par défaut en premier). Après les constructeurs, insérez les accesseurs et les mutateurs de la classe. Faites suivre les accesseurs et les mutateurs par toutes vos méthodes.

1 Création d'un planning initial

Si jamais vous n'aviez pas terminé la première partie (TP 5), il serait grand temps de le faire ... En effet, avant de passer à l'optimisation du planning de livraison, il faudrait déjà avoir une première solution initiale, comme décrit à la fin du TP 5.

2 Optimisation du planning

Nous avons mis tout en place pour pouvoir sortir des plannings de livraison sur les instances que Monsieur Houblon nous donnera. Par contre, l'algorithme qui calcule le planning n'est pas du tout optimisé. Nous voulons donc améliorer l'aspect algorithmique de notre outil.

2.1 Réduction du nombre de tournées

L'optimisation basique que nous avons écrit lors du dernier TP ajoute un client dans la tournée courante si la capacité résiduelle du véhicule est respectée. Sinon, une nouvelle tournée est créée pour accueillir le client. Nous notons ici que lorsque l'insertion d'un client dans une tournée échoue, la tournée n'est plus jamais prise en considération. Pourtant, le véhicule associé pourrait avoir de la place pour livrer d'autres clients (qui auraient une demande plus faible).

Question 1. Ajoutez dans la classe **Planning** une méthode **planificationMinTournee(Depot depot, Set<Client> clients)** qui cherche d'abord à insérer un client dans une des tournées existantes. Seulement si aucune tournée ne dispose de la capacité nécessaire pour livrer les caisses de bière requises par le client, alors une nouvelle tournée est créée. Notez que le calcul de la distance totale dans la classe **Planning** pourra alors être réalisé lorsque tous les clients ont été ajoutés dans une tournée. Testez le nouvel algorithme et comparez-le avec l'optimisation basique (en termes de nombre de tournées créées et de distance parcourue).

2.2 Meilleure insertion

Dans cette seconde section, nous allons voir l'algorithme de la meilleure insertion (*best insertion* en anglais). L'idée de cet algorithme est la suivante : étant donné un ensemble de tournées et un client, ce dernier est inséré dans la meilleure position possible dans les tournées existantes. Dans notre cas, la meilleure insertion est celle qui permet de réaliser le plus petit détour (tout en respectent la capacité du véhicule) pour livrer le client à ajouter. Pour ce faire, nous devons parcourir toutes les tournées de notre planning. Pour chaque tournée, on vérifie d'abord que l'insertion du client peut être réalisée, i.e. que la capacité du véhicule est respectée. Si tel est le cas, on cherche la position d'insertion du client dans la tournée qui permet de réaliser le plus petit détour possible. Ainsi, on n'insère plus le client en fin, mais on teste l'insertion du client juste après le dépôt et

après chacun des clients déjà présents dans la tournée. La position qui permet de réaliser le plus petit détour est retenue. Jusque là on fait seulement des tests, mais le client n'a été inséré dans aucune tournée. Le client est finalement inséré dans la tournée qui permet de réaliser le plus petit détour (et dans cette tournée, il est inséré à la position qui permet le plus petit détour).

Question 2. Créez dans le paquetage **planning** une classe **BestInsertion** avec au minimum les trois attributs suivant : un de type **Tournee**, un de type **double** et un de type **int** pour mémoriser respectivement la tournée, la longueur du détour et la position dans la tournée associée à la meilleure insertion. Ajoutez les constructeurs, accesseurs et mutateurs nécessaires. Il pourra être utile d'ajouter un constructeur par défaut (tournée **null** et distance du détour infinie).

Question 3. Ajoutez dans la classe **Tournee** une méthode **public BestInsertion calculerBestInsertion(Client nouveauClient)**. Cette méthode calcule la meilleure insertion du **nouveauClient** dans la tournée courante (sans réaliser cette insertion) et renvoie une instance de la classe **BestInsertion** avec les informations associées à la meilleure insertion. Si l'insertion n'est pas réalisable, il faudra renvoyer **null** ou bien une instance "par défaut" de **BestInsertion**.

Question 4. Ajoutez dans la classe **Planning** une méthode **planificationBestInsertion(Depot depot, Set<Client> clients)**. Cette méthode parcourt les tournées actuelles du planning et, pour chacune, fait appel à la méthode **calculerBestInsertion**. Puis, elle réalise effectivement l'insertion à l'endroit qui permet de faire le plus petit détour. Si un client n'a pu être inséré dans aucune tournée, alors on crée une nouvelle tournée.

Question 5. Ajoutez dans la classe **Routage** une méthode **public static void comparerMethodes(String fichierEntree)**. La méthode lit le fichier d'entrée, initialise toutes les routes du réseau routier (avec un appel à la méthode **initialiserRoutes()**), puis elle fait appel aux trois méthodes de planification que vous avez écrit. Enfin, le planning fourni par chaque algorithme d'optimisation est écrit dans un fichier différent. Le fichier sera nommé de manière automatique. Les trois fichiers de sortie pourront être mis dans des répertoires différents.

Question 6. Testez vos algorithmes de planification sur chacun des fichiers de test disponibles sur Moodle.

Comparez vos résultats. Sont-ils conformes à vos attentes ?

Question 7. Il est fort probable que les solutions obtenues puissent être améliorées. Proposez de nouveaux algorithmes de planification afin d'améliorer vos résultats.