INFO-F103 – Algorithmique 1
Projet 1 : *Planning de livraison*

Jérôme De Boeck

Année académique 2015–2016

Enoncé

Vous êtes en charge d'une société de livraison et devez trouver un itinéraire dans un réseau routier qui permet de livrer un ensemble de clients. Chaque client à une heure maximum de livraison. Avant de pouvoir livrer un client, le camion de livraison doit être passé par le dépôt attribué à ce client pour aller chercher la marchandise qui lui est destinée. Les marchandises étant placées l'une derrière l'autre dans le camion, elles doivent être déchargées dans l'ordre inverse de l'ordre de chargement. Un client ne pourra donc être livré que si sa marchandise est la dernière marchandise à avoir été embarquée.

Par simplicité, nous considérons que le camion n'a pas de limite de chargement et pourrait donc, si nécessaire, passer par tous les dépôts et charger toutes les marchandises avant de livrer tous les clients, en respectant l'ordre de déchargement. Nous considérons aussi que les chargements aux dépôts et les déchargements chez les clients prennent tous un temps de 5 minutes. Le camion peut passer par un dépôt sans restriction et a le choix d'effectuer ou non un chargement de marchandise lors de son passage. Par contre pour passer chez un client, la marchandise destinée à ce client doit être la dernière embarquée pour que celui-ci puisse être livré ou celui-ci doit déjà avoir été livré. Le camion ne peut donc pas passer chez un client si celui n'a pas été livré et que le camion ne peut pas le livrer dans l'immédiat.

Pour limiter les possibilités de recherche, le camion peut passer au maximum deux fois par chaque lieu. Le point de départ du camion est considéré comme étant visité une première fois au début de la tournée.

L'itinéraire à trouver doit respecter l'heure de livraison de chaque client, l'ordre de déchargement des marchandises en fonction de l'ordre de chargement, le nombre de passages et prendre le moins de temps possible. L'itinéraire se termine à la fin du déchargement du dernier client, le camion ne doit pas revenir au point de départ. L'heure de livraison d'un client tient compte du temps de déchargement. Par exemple, si un client doit être livré à 10h, que le camion arrive à 9h56 et que sa livraison est donc finie à 10h01, l'heure de livraison n'est pas respectée.

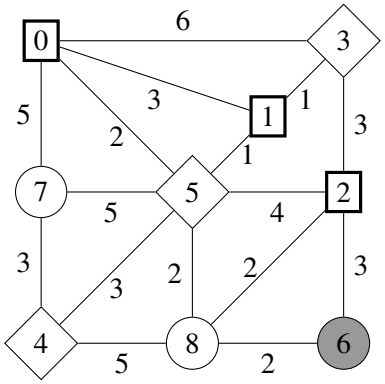
Implémentation

Le réseau routier comprend n clients (numérotés de 0 à $n-1$), n dépôts (numérotés de n à $2n-1$) et un ensemble de carrefours supplémentaires dont le point de départ du camion (numéroté $2n$). Les clients, dépôts et carrefours seront d'une manière plus générale appelés des *lieux*, il y en a m au total. Le dépôt du client i est le dépôt $i+n$. Les temps de trajet entre les différents lieux de ce réseau sont repris dans la matrice `time` où

$$\texttt{time}[i][j] = \begin{cases} \text{temps nécessaire pour aller de } i \text{ à } j, & \text{si } i \text{ et } j \text{ sont voisins} \\ -1, & \text{sinon} \end{cases}$$

où $i, j \in \{0, ..., m-1\}$

Par exemple dans le réseau de la figure 1 avec $n = 3$: il y a trois clients, les dépôts des clients 0,1 et 2 sont respectivement les dépôts 3,4 et 5, le point de départ est le carrefour 6 et il y a deux carrefours supplémentaires numérotés 7 et 8.



$$\texttt{time}= \begin{pmatrix} -1 & 3 & -1 & 6 & -1 & 2 & -1 & 5 & -1 \\ 3 & -1 & -1 & 1 & -1 & 1 & -1 & -1 & -1 \\ -1 & -1 & -1 & 3 & -1 & 4 & 3 & -1 & 2 \\ 6 & 1 & 3 & -1 & -1 & -1 & -1 & -1 & -1 \\ -1 & -1 & -1 & -1 & -1 & 3 & -1 & 3 & 5 \\ 2 & 1 & 4 & -1 & 3 & -1 & -1 & 5 & 2 \\ -1 & -1 & 3 & -1 & -1 & -1 & -1 & -1 & 2 \\ 5 & -1 & -1 & -1 & 3 & 5 & -1 & -1 & -1 \\ -1 & -1 & 2 & -1 & 5 & 2 & 2 & -1 & -1 \end{pmatrix}$$

FIGURE 1 – Exemple de réseau et de matrice `time` correspondante

Le camion de livraison démarre toujours à 8h qui sera le temps 0. Une liste `deliveryTime` contient les temps maximums de livraison exprimés en minutes à partir du temps 0. `deliveryTime[i]` représente l'heure maximum de livraison du client `i`. Si `deliveryTime[0]` = 150 alors le client 0 devra être livré au plus tard à 10h30 (en comptant le déchargement). Les données `n,m,deliveryTime,time` seront contenues dans un fichier dont le format et un exemple sont donnés en annexe.

Votre algorithme de résolution doit être contenu dans une classe `Planning` et utiliser une méthode de backtracking `solve()`. Utilisez des méthodes de branch and bound pour réduire le nombre d'itérations.

La classe `Planning` doit avoir comme attributs

- les informations d'une instance `n`, `m`, `deliveryTime`, `time`
- un entier `totalTime`, initialisé à 0, qui contiendra le temps total du trajet optimal

- une liste `parcours` qui contient les numéros des lieux visités du trajet optimal
- une liste `actions` qui contiendra pour chaque entrée de `parcours` les informations des actions effectuées à ce lieu
- un entier `count`, initialisé à 0, qui contiendra le nombre total d'appel à la méthode `solve()` lors de la résolution

Si aucun trajet n'est trouvé, ces variables conserveront leurs valeurs initiales. Il peut bien sur y avoir des attributs de travail supplémentaires dans la classe.
La classe `Planning` doit avoir un constructeur qui reçoit en paramètre le nom d'un fichier de données, initialise l'instance avec les données de ce fichier, appelle la méthode `solve()` et affiche la meilleure solution trouvée via une méthode `printRes()`. Cette méthode affiche les différentes actions de la pile `actions` ligne par ligne, le temps total et le nombre d'itérations du compteur `count` via une méthode `printRes()`. Le format des actions est décrit en annexe.

La classe `Stack` disponible sur l'uv peut être utilisée pour modéliser des variables de travail, elle devra être implémentée dans le même fichier que la classe `Planning`. Utilisez au mieux les structures de données vues au cours.

Pour exécuter votre algorithme à partir d'un interpréteur python sur une instance contenue dans un fichier `data.txt` (qui est dans le même dossier), l'utilisateur devra taper :

```
>>> from projet1 import *
>>> Planning("data.txt")
```

Un exemple de résolution est donné en annexe et des instances de test seront fournis sur l'UV ainsi que les solutions correspondantes. Vérifiez que votre programme fonction en l'exécutant via le terminal (et non dans un framework) avant de le remettre.

Consignes générales

Tout votre code doit être contenu dans un seul fichier **projet1.py**. Veuillez respecter le nom, la casse et le format des variables, de la classe et du fichier.
Nous vous demandons de

- créer localement sur votre machine un répertoire de la forme NOM_Prenom (exemple :DUPONT_Jean) dans lequel vous mettez le fichier à soumettre
- compresser ce répertoire via un utilitaire d'archivage produisant un .zip
- de soumettre le fichier archive .zip, et uniquement ce fichier, sur l'UV

Les consignes générales des projets en BA1 sont celles reprises dans le document `http://uv.ulb.ac.be/pluginfile.php/524355/mod_resource/content/1/consignes_projets.pdf`.
La version électronique est à remettre pour le **lundi 21 mars 2016** à 10h, la version papier pour la même date entre 10h et 12h au secrétariat étudiant, aucun retard n'est toléré. Tout manquement aux consignes ou retard sera sanctionné directement d'un **0/10**.

Annexes

Fichier de données

Les fichiers de données sont écrits au format suivant (`m` est le nombre de lieux dans le réseau, `n` est le nombre de clients) :

```
n
m
delivery-0 delivery-1 ... delivery-n-1
time[0][0] time[0][1] ... time[0][m]
time[1][0] time[1][1] ... time[1][m]
time[m][0] time[m][1] ... time[m][m]
```

Le fichier reprenant les données de la figure 1 avec trois clients à livrer respectivement avant 8h55, 8h50 et 8h40 sera le suivant :

```
3
9
55 50 40
-1 3 -1 6 -1 2 -1 5 -1
3 -1 -1 1 -1 1 -1 -1 -1
-1 -1 -1 3 -1 4 3 -1 2
6 1 3 -1 -1 -1 -1 -1 -1
-1 -1 -1 -1 -1 3 -1 3 5
2 1 4 -1 3 -1 -1 5 2
-1 -1 3 -1 -1 -1 -1 -1 2
5 -1 -1 -1 3 5 -1 -1 -1
-1 -1 2 -1 5 2 2 -1 -1
```

Pour rappel les dépôts des clients aux sommets 0, 1 et 2 sont respectivement les dépôts 3, 4 et 5 et le carrefour de départ sera le 6.

Output

A la fin de la construction d'une instance de la classe `Planning` :

- si aucun trajet ne permet de livrer tous les clients dans le temps imparti, afficher un message d'erreur
- si un itinéraire est trouvé, afficher les entrées de `actions`. Une action doit contenir le numéro du lieu, l'heure d'arrivée, le type de lieu, l'action effectuée s'il y en a une avec son heure de fin ainsi que le retard s'il s'agit d'une livraison. Afficher ensuite le temps total de la tournée `totalTime` et nombre d'itérations `count`.

  Un exemple d'exécution sur le fichier `data.txt` représentant la figure 1 :

  ```
  >>> from projet1 import *
  >>> planning = Planning("data.txt")
  6 : 8h00, carrefour
  8 : 8h02, carrefour
  ```

```
5 : 8h04, depot du client 2, chargement fini a 8h09
4 : 8h12, depot du client 1, chargement fini a 8h17
5 : 8h20, depot du client 2
1 : 8h21, client 1, dechargement fini a 8h26
3 : 8h27, depot du client 0
2 : 8h30, client 2, dechargement fini a 8h35
3 : 8h38, depot du client 0, chargement fini a 8h43
1 : 8h44, client 1
0 : 8h47, client 0, dechargement fini a 8h52
Temps total : 52
Nombre d'itérations : 5289
>>> planning.totalTime
52
>>> planning.count
5289
>>> planning.parcours
[6,8,5,4,5,1,3,2,3,1,0]
```