Institut Villebon - Georges Charpak

Projet : mon_via_navigo

1 Objectif du projet

La R.A.T.P., la S.N.C.F. et la région Ile-de-France (à travers ViaNavigo) proposent aux usagers des transports en commun un service interactif permettant de trouver le plus court trajet entre deux points d'un réseau de transport laissés au choix de l'utilisateur.

L'objectif du projet sera de réaliser une version (simplifiée) d'un tel logiciel, et de réfléchir aux stratégies les plus efficaces pour résoudre des problèmes du même type.

La méthode générale pour réaliser un tel logiciel est tout simplement d'appliquer l'algorithme de Dijkstra à un graphe valué orienté dont les sommets sont les stations ou arrêts, les arêtes symbolisant un lien entre deux stations ou arrêts d'une ligne et enfin les valuation des arêtes étant le temps moyens de trajets entre les stations représentées par les sommets de l'arête.

Plus précisément,

- chaque sommet du graphe correspond à une station pour une ligne donnée (par exemple, République [ligne 3] et République [ligne 5] sont deux sommets différents).
- à chaque sommet est associé la position de la station sur une carte (échelle : 1 / 25000).
- à chaque sommet est associé le nom de la station (sous forme de chaîne de caractères).
- deux sommets forment un arc orienté si le métro, un bus ou un moyen de transport public, relie directement les stations correspondantes : notons que le graphe est orienté en raison de quelques "sens uniques" (par exemple du côté de la porte d'Auteuil, pour ce qui concerne Paris).
- un arc est valué par le temps estimé de son trajet en secondes.

Votre programme python s'exécutera suivant la syntaxe suivante :

./mon_via_navigo station_1 station_2 [heure]

où `station_1` et `station_2` sont deux stations ou arrêts du réseau de transport en commun. `heure` est un paramètre optionnel qui sera expliqué plus tard

2 Des classes préliminaires

2.1 La classe `Station`

Une classe `Station` sera crée et permettra de représenter un sommet du réseau de transport public. Ses attributs seront le nom de la station, appelé `name`, sa position `position` qui sera un couple de flottants ainsi que son appartenance à une ligne donné par l'attribut `in_line` qui sera une chaine de caractère initialement vide.

Cette classe devra probablement posséder une implémentation des méthodes suivantes :

- `__eq__(self, station)` permettant de décider si deux stations/arrêts sont identiques : elle renverra donc un booléen.
- `__lt__(self, station)` permettant de décider laquelle des deux stations parmi `self` et `station` est inférieure (strictement) à l'autre pour l'ordre lexicographique (appliqué sur le nom de la station, puis sur le nom de la ligne d'appartenance en cas de même nom.
- `__hash__(self)` renvoyant immédiatement `hash(self.name + self.in_line)`
- `__str__(self)` renvoyant une représentation textuelle (*i.e.* une chaine de caractères) raisonnable permettant d'identifier la station

2.2 La classe `Line`

Une classe `Line` sera crée et permettra de représenter une ligne de transport public. Puisqu'une véritable ligne de transport public peut comporter des sens uniques, celle-ci sera orientée.

Notons que la modélisation est légèrement différente de ce que l'on appelle couramment une ligne de transport en commun. Généralement, une ligne n'est pas considérée comme orientée ; lorsque le besoin d'une orientation se fait ressentir, nous utilisons les terminus pour indiquer le sens de direction. Ainsi, pour représenter une ligne (dont les terminus sont les stations A et B) dans le sens courant, nous aurons à notre disposition deux objets de type `Line` : un pour symboliser la direction de A vers B, un autre pour la direction de B vers A.

Les attributs de cette classe seront :

- `stations` qui sera une liste des stations que la ligne comporte
- `mode` qui désignera le type de transport utilisé (bus, métro, RER, vélib, ...)
- `name` qui sera une chaine de caractères donnant un nom à la ligne de transport.

Lors de l'initialisation d'un objet de cette classe, il faudra ne pas oublier de modifier l'appartenance des stations à cette ligne, *i.e.* modifier l'attribut `in_line` de chacune de ses stations.

Enfin, une représentation textuelle, *i.e.* une méthode `__repr__(self)`, renvoyant une chaine de caractères pourra être utile.

2.3 La classe `PublicTransportationNetwork`

Une classe `PublicTransportationNetwork` sera crée et permettra de représenter un réseau de transport public, c'est-à-dire un ensemble de lignes de transport. Notons qu'il s'agit du cœur même du logiciel Vianavigo.

Les attributs de cette classe seront au minimum :

- `name` permettant de connaitre le nom du réseau.
- `scale` permettant de connaitre l'échelle à laquelle le réseau est représenté en machine et donc de savoir utiliser correctement l'attribut `position` de la classe `Station`
- `lines` représentant l'ensemble des lignes de transport du réseau.

Lors de la création d'un objet `PublicTransportationNetwork`, l'attribut `lines` sera vide. Il sera complété au fur et à mesure grâce à une méthode `add_a_line(self, line)`.

Après avoir été crée, un réseau pourra être sauvegardé en machine ; un réseau pourra donc aussi être chargé. Cela signifie que des méthodes `save(self, file_name)` et `load(self, file_name)` seront implémentées. Elles permettront notamment de charger le réseau du métro parisien.

Les fichiers de sauvegarde d'un réseau seront des fichiers de type `.csv` [1]dont le séparateur est `:`. Chaque ligne de transport sera sauvegardé dans son propre fichier ; les informations du réseau seront aussi sauvegardées dans un fichier.

Enfin, il faudra aussi être capable de représenter en console un réseau, donc implémenter une méthode `__str__(self)`.

3 Niveau de réalisation du projet

3.1 Niveau Bryan : "Bryan is in the kitchen"

Il s'agit dans un tout premier temps d'implémenter les classes décrites précédemment, de sorte qu'elles fonctionnent correctement.

Notamment, vous effectuerez des tests pour chaque méthode écrite qui feront parti du rendu.

3.2 Niveau Nicky Larson

Dans un second temps, il faudra être capable de créer un réseau virtuel de transport pour pouvoir tester plus tard le programme. Pour cela, vous écrirez les fonctions suivantes :

- `create_circular_line(line_name, stations_name, center, radius, mode`, qui permet de créer une ligne circulaire dans votre ville fictive dont les noms des stations (et donc le nombre) sont données par `stations_name` et sont réparties équitablement sur le cercle de centre `center` et de rayon `radius`.
- `create_straight_line(line_name, stations_names, from_x, from_y, to_x, to_y, mode)`, qui permet de créer une ligne de transport qui soit une ligne droite parfaite passant par les points `(from_x, from_y)` et `(to_x, to_y)` et dont les stations sont réparties équitablement sur la droite et leur nom sont données par `stations_name`.

Ces deux fonctions renverront un objet de type `Line`.

Vous créerez notamment le réseau de transport en commun de deux villes. Ceux-ci seront évidemment sauvegardés et le rendu contiendra une réprésentation graphique sommaire de ces plans.

Des tests seront toujours écris pour chaque méthode et feront parti du rendu.

3.3 Niveau Sherlock Holmes : "It was the simplicity itself, Watson"

L'objectif ici est d'être capable de déterminer le plus court trajet entre deux points du réseau de transport et d'évaluer sa durée.

La classe `PublicTransportationNetwork` contiendra une méthode fondamentale `create_graph(self)` permettant de construire le graphe associé au réseau de transport public. Ses sommets seront l'ensemble des stations/arrêts, ses arêtes symboliseront le fait que deux stations sont accessibles en une station par un bus, un RER, un métro, $\cdots$ La valuation des arêtes désignera le temps moyen

[1] `csv` signifie "comma separated values". Pour plus d'information sur ce format, on pourra consulter la page wikipédia Comma-separated values.

pour parcourir un trajet élémentaire entre deux stations consécutives d'une même ligne, ou alors entre correspondance.

Pour déterminer la valuation des arêtes du graphes, on utilisera la fonction `duration(station_1, station_2, scale, mode)` renvoyant le temps de trajet entre les stations `station_1` et `station_2`, où :

- `station_1` et `station_2` appartiennent à la même ligne du réseau, ou se doivent d'être voisine dans le graphe représentant le réseau. Dans le cas contraire, une exception sera levée
- `scale` désigne l'échelle de l'implémentation machine du réseau
- `mode` est le mode de transport utilisé. Celui-ci devra être compatible avec le cheminement demandé dans le graphe représentant le réseau de transport. Dans le cas contraire, une exception sera levée.

Pour calculer le temps de parcours, on utilisera les vitesses commerciales moyennes suivantes :

Type de transport	Vitesse commerciale
RER	40 km/h
Métro	25 km/h
Tramway	22 km/h
Bus	14 km/h
piéton	4.5 km/h

Déterminer le trajet le plus court en temps reviens alors à appliquer l'algorithme de Dijkstra au graphe construit, et nécessite donc d'avoir une classe `Graph` opérationnelle...

A ce stade, votre programme doit permettre de retrouver le comportement suivant :

>>> ./mon_via_navigo Blanche Orsay_ville
Ligne 2 : Blanche −−> La Chapelle
Correspondance : La Chapelle −−> Gare du nord
RER B : Gare du nord −−> Orsay ville

Duree estimee : 1h 25min

3.4 Niveau Barney Stinson : "It's gonna be legend-... wait for it... dairy!"

Désormais, il s'agit de rajouter un paramètre : malheureusement, les bus, les métros ou les trains ne circulent pas dès qu'un utilisateur claque des doigts...

Faites en sorte de pouvoir prendre en compte :

- des fiches d'horaires sur chaque ligne du réseau de transport (celles-ci pourront être sauvegardées sous forme de fichier `.csv`)
- le temps d'attente entre correspondances ;

Votre programme devra permettre de prendre en compte le paramètre optionnel `heure` indiquant l'heure de départ souhaitée par l'usager. Votre programme doit alors permettre de retrouver le comportement suivant :

>>> ./mon_via_navigo Blanche Orsay_ville 6h15
6h20 : Ligne 2 : Blanche −−> La Chapelle
6h35 : Correspondance : La Chapelle −−> Gare du nord
6h50 : RER B : Gare du nord −−> Orsay ville
Arrivee estimee a 7h 59

3.5 Niveau Martin Luther King : "I have a dream"

Pour ce dernier niveau, il s'agit non plus de calculer le plus court trajet, mais de tenter de trouver les trois trajets qui devraient être parmi les plus court et de les présenter comme suit :

>>> ./mon_via_navigo Blanche Orsay_ville 6h15
Trajet le plus court :
6h20 : Ligne 2 : Blanche −−> La Chapelle
6h35 : Correspondance : La Chapelle −−> Gare du nord
6h50 : RER B : Gare du nord −−> Orsay ville
Arrivee estimee a 7h 59

Autre proposition :
6h20 : Ligne 2 : Blanche −−> Place de Clichy
6h25 : Ligne 13 : Place de Clichy −−> Saint−Lazare
6h35 : Ligne 14 : Saint−Lazare −−> Chatelet
6h52 : RER B : Chatelet −−> Orsay ville
Arrivee estimee a 7h 59

Autre proposition :
6h20 : Velib : Blanche −−> Denfert Rochereau
7h03 : RER B : Denfert−Rochereau −−> Orsay ville
Arrivee estimee a 8h12

4 Condition de rendu du projet

Ce projet sera à rendre par mail à `olivier.bouillot@villebon-charpak.fr`, au plus tard le mercredi 26 janvier à 8h00. Il sera à réaliser en trinôme. Un rapport de projet sera rendu avec les fichiers sources de votre projet.

Une soutenance aura ensuite lieu le vendredi 29 au cours de laquelle vous devrez :

- faire une démonstration de votre projet ;
- démontrer la maitrise intégrale de votre code ;
- présenter les améliorations que vous avez codées ;
- répondre aux questions du jury.

Pour une telle soutenance, il n'est pas nécessaire de préparer de présentation Beamer, PowerPoint ou équivalent, ni de suivre le contenu du rapport de projet. Ce dernier servira à apprécier tout autant que la soutenance la qualité de la réalisation du projet.

4.1 Les trinômes

Les trinômes pour réaliser le projet sont les suivants :

Abdoulakime	Thirusittampalam	Vong
Albrand	Herat	Kechiche
Barsom	Bezieau	Solier
Beltran	Geffroy	Malpart
Chanau	Chazelas	Monnier
Erard	Godet	Varin
Gosselin	Kadogami	Magni
Lavrat	Mariejoseph	Sonko

Ces groupes ont été conçus pour garder les bons binômes de T.P. autant que possible et aider les binômes de T.P. ayant plus de difficultés.

4.2 Contenu du projet et notation

Vous veillerez à ne pas modifier les noms donnés dans l'énoncé, car une partie de votre projet sera corrigé automatiquement : si vous ne respectez pas cette consigne, vous aurez alors 0 à cette partie, car aucun des tests que votre code subira ne passera...

Néanmoins, pour les parties plus libres où vous aurez à concevoir vous-même l'architecture et l'algorithmique, vous choisirez vous même les noms de variables. Ceux-ci se devront d'être clair et limpide, mais en anglais.

Bien sur, vous n'oublierez pas de documenter votre code (commentaires + docstring). L'idéal serait même d'avoir des doctests (*i.e.* des exemples d'utilisation de la méthode ou fonction situés à l'intérieur de sa docstring) : voir le paragraphe sur les doctests de "Les docstrings en Python : les docstrings". Pour un projet de cet envergure, il peut être tout à fait judicieux, dans les docstrings, d'avoir un champ pour le nom de l'auteur du code, et un autre pour la date de dernière modification et une description de ces dernières modifications.

Vous respecterez des règles d'hygiènes correctes de codage : cela sera hautement pris en compte dans la notation. En effet, des tests automatiques seront effectués pour vérifier si les règles du PEP 0008 sont respectées. Pour mémoire, voir l'énoncé du TP1 ainsi que "PEP 0008 – Style Guide for Python Code".

Enfin, a priori, une et une seule note sera attribuée au groupe, mais il sera tout à fait possible que certains membres du groupe aient finalement une note autre, en fonction de leur capacité à répondre aux questions et de leur investissement.

Pour résumer, il est fort probable que la note finale soit produit suivant les critères suivants :

Hygiène typographique du code	5%
Qualité des docstrings	10%
Qualité des tests	10%
Réussite dans le projet	25%
Qualité du rapport de projet	25%
Soutenance orale	25%

Pour avoir la moyenne, de manière générale, il faut au moins avoir atteint le niveau “Sherlock” et fait en sorte qu'il fonctionne correctement.

4.3 Rapport de projet

Votre compte-rendu devra suivre la structure suivante :

I. expliquer en détails comment faire fonctionner votre application, notamment avec des exemples d'utilisations simples à reproduire : il s'agit d'une documentation utilisateur ;

II. expliquer en détail le fonctionnement interne de votre application : il s'agit d'une documentation technique ;

III. indiquer les difficultés que vous avez rencontrées et les différentes solutions mises en œuvre pour les contourner, notamment il faudra rendre intelligible vos choix d'implémentation (si vous aviez le choix entre plusieurs solutions, vous indiquerez aussi les raisons de votre choix ; si vous avez recopié du code et ne comprenez pas parfaitement son fonctionnement, c'est ici qu'il vous faudra le mentionner, ...) ; en cas d'absence de solutions, vous l'indiquerez également ;

IV. détailler les options personnelles de fonctionnement que vous avez implémentées, ainsi que celles que vous auriez aimé implémenter si le temps vous l'avez permis.

4.4 Rendu du projet

Votre mail d'envoi aura pour sujet :

[UE 5i4] Rendu de projet `prenom_1` `prenom_2` `prenom_3`

où `prenom_1`, `prenom_2` et `prenom_3` seront bien entendu remplacé par les prénoms des membres du trinôme. Il contiendra une et une seule pièce jointe qui sera une archive nommée

`Projet_-_nom_1_nom_2_nom_3.zip`

où `nom_1`, `nom_2` et `nom_3` correspondent aux noms de familles des membres du groupe et seront :

- classé par ordre alphabétique.
- écris avec une première lettre en majuscule, les autres étant en minuscule.

Si ces conditions ne sont pas remplies, une sanction sur la note finale sera appliquées : un point de moins par condition non remplie (si les trois noms sont mal formatés, cela comptera pour trois conditions, donc trois points de perdu[2])

[2]Heureusement, vous êtes toujours tous présent en cours, donc vous avez tous gratuitement un bonus de deux points !