Intelligence Artificielle : Logique et Contraintes - Devoir n° 1

Organisation de visites à domiciles pour un cabinet infirmier

Consignes à respecter pour votre devoir

Ce devoir doit être réalisé **en binôme**.
Vous rendrez ce devoir à la fois sous la forme d'un compte rendu papier, à déposer au secrétariat, et d'une archive au format `.tgz` dont le nom sera **impérativement** de la forme `Devoir_M1-App_IALC_Nom1_Nom2.tgz` (où Nom1 et Nom2 sont les noms respectifs des deux membres du binôme, dans l'ordre alphabétique).
Le sujet du mail devra être : `[M1-App] : Devoir IALC Nom1 Nom2`
L'archive devra contenir :

- un répertoire `doc`, avec une version **en pdf**, du compte rendu papier nommé `cr_devoir_ialc_Nom1_Nom2.pdf`, et qui devra détailler votre modélisation, vos choix de mise en oeuvre et votre analyse des résultats obtenus sur les différentes instances ($\approx$ 10 pages).
- un répertoire `src` contenant le code source de votre devoir, i.e. les fichiers de modèles OPL réalisés et le code de script. Vous penserez à mettre les noms de votre binôme en commentaire en tête de chaque fichier source.
- un répertoire `lib` avec les librairies qui vous ont été fournies
- un répertoire `data` avec les données qui vous ont été fournies et les fichiers `.txt` décrivant les données de vos instances.
- un répertoire `instances` avec les fichiers `.dat` décrivant vos instances (que vous pouvez structurer à votre guise)
- un répertoire `resultats` où devront être enregistrés les fichiers résultats

Votre archive est à rendre par email **au plus tard le vendredi 06/04/19 à 16h00 (strict)**. La version papier doit être déposée au secrétariat ce même jour. Aucun envoi ne sera pris en compte au delà. Prenez donc vos dispositions pour être certains d'avoir rendu au moins une version de votre devoir avant cette limite et d'anticiper sur tout problème de transport, panne de réseau, machine, etc... Vous pouvez éventuellement envoyer une version modifiée de votre devoir jusqu'à la limite fixée. Vous recevrez un accusé de réception dès que possible. Merci de réagir rapidement (dans les 24h) si ce n'est pas le cas. Afin limiter les risques de mails égarés, merci d'utiliser votre adresse `prenom.nom@u-psud.fr` pour l'envoi et pensez toujours à bien mettre votre binôme en copie du mail.

Enfin, il vous est demandé **un travail personnel**. Sachez que tous les codes seront attentivement vérifiés, comparés et **toute copie de code entre binômes sera sévèrement sanctionnée.**

Remarque préalable : ne vous laissez pas impressionner par la longueur de l'énoncé, l'objectif est de lever au maximum toutes les ambiguïtés et de vous guider au mieux.

I) Le contexte du problème

Sophie travaille avec quelques collègues comme infirmière libérale dans un cabinet en milieu rural. Chaque jour elle doit se rendre successivement chez différents patients à l'aide de sa voiture, pour y accomplir différents actes, qui peuvent être soit de nature médicale (e.g. faire une prise de sang, changer un pansement, enlever des fils de suture,...), soit des actes de soin (faire une toilette légère, complète,..). Pour certains patients, il peut s'agir d'un acte simple mais bien souvent, il s'agit d'une

combinaison d'actes (notamment pour les malades âgés en situation de dépendance). Certains patients nécessiteront une visite unique, d'autres auront besoin de visites régulières, parfois même plusieurs dans une même journée. La durée de chaque visite varie en fonction des actes dispensés.

Sophie apprécie son métier parce que c'est un métier de contact, qui lui permet de rencontrer des personnes très différentes mais aussi parce que chaque journée est différente de la précédente. Cependant elle commence toujours sa journée en passant d'abord par son cabinet, pour s'assurer qu'il n'y a pas d'urgence ni d'annulations. Ensuite elle va visiter ses différents patients. Après sa dernière visite, elle doit de nouveau repasser à son cabinet, pour y effectuer ses *transmissions* à la Sécurité Sociale (une tâche qui lui prend en moyenne 5mn par tranche de 3 patients visités dans la journée).

Certaines journées peuvent s'avérer plus longues que d'autres. L'incertitude liée à l'exercice en mode libéral fait qu'on ne sait jamais de quoi demain sera fait et il est souvent difficile de refuser la prise en charge d'un nouveau patient. En même temps, elle est sait que pour bien exercer son métier, elle doit rester vigilante et concentrée et se garder de tomber dans le surmenage (ce qui nécessite de bien mesurer l'impact de chaque nouvelle prise en charge). Pour cela, il est important de bien s'organiser. Sophie a remarqué que certain jours elle passe vraiment beaucoup de temps dans sa voiture. Or si le temps passé chez les patients reste incompressible, l'ordre dans lequel elle effectue ses visites peut avoir un impact considérable sur la longueur de ses journées de travail. Son intuition lui suggère qu'il est préférable d'enchaîner des visites chez des patients dont les domiciles son proches. Mais certaines visites peuvent être assorties de contraintes temporelles, restreignant les moments où elles peuvent avoir lieu. Il n'est donc pas toujours aisé de savoir comment organiser au mieux ses visites.

Pour optimiser ses trajets, Sophie souhaite exploiter des données libres collectées dans la base de donnée géographique d'OpenStreetMap (`https://www.openstreetmap.org`) et tout particulièrement celles du projet OSRM (`http://project-osrm.org`), qui offre différents services permettant notamment d'obtenir les temps de parcours entre un ensembles de points repérés par leurs coordonnées GPS. Sophie n'a cependant pas l'intention de perdre du temps à rechercher les coordonnées GPS de chacun ses patients. Elle souhaite une solution pratique lui permettant planifier au mieux toutes ses visites.

Elle a donc imaginé un format simple pour décrire toutes les informations qui lui semblent utiles pour organiser ses visites sur une journée. Elle sait qu'elle doit notamment décrire les soins qu'elle doit administrer à chacun de ses patients et, pour chacun d'eux, leur adresse, leurs (in)disponibilités, etc. Elle compte sur vous pour concevoir différents modèles aptes à lui indiquer la meilleure façon d'organiser ses visites en fonction des critères du moment.

Le format qu'elle a retenu permet de décrire chaque élément d'information par une ligne débutant par un mot particulier, suivi d'une suite de paramètres séparés par des espaces et/ou tabulations et répondant à une syntaxe précise. Les différentes informations peuvent être indifféremment regroupées dans un même fichier ou réparties dans plusieurs fichiers différents, au format texte. Il est à noter qu'aucune hypothèse n'est faite sur l'ordre dans lequel ces informations sont décrites.

Les différentes types d'informations acceptés sont décrits à l'aide de la syntaxe suivante (où chaque expression entre crochets `<...>` représente un paramètre) :

- `journee` <$InfJ$> <$SupJ$>
  où <$InfJ$> et <$SupJ$> décrivent tous deux un horaire de la forme xx`h` ou xx`h`yy, où xx (resp. yy) est un entier décrit sur deux chiffres entre `00` et `23` (resp. `59`) et correspondant au nombre d'heures (resp. de minutes) de l'horaire (e.g. `07h`, `20h30`, ...). Cette information décrit la plage horaire de la journée, dans laquelle devront s'effectuer toutes les visites considérées. Si une telle ligne ne figure pas dans la description de l'instance à traiter, on supposera par défaut que la journée commence à `07h00` et se termine à `21h00`.
- `infirmiere` <Nom> <Inf> <Sup>
  où <Nom> est le nom d'une infirmière et ou $<Inf>$ $<Sup>$ sont facultatifs et décrivent l'intervalle horaire durant lequel l'infirmière travaille. S'ils sont omis, on retiendra comme valeurs $<InfJ>$ $<SupJ>$
- `indisponibilite` <Nom> <Inf> <Sup>
  où <Nom> est le nom d'un patient ou d'une infirmière et <Inf> <Sup> défini un fenêtre temporelle durant laquelle un patient ne peut être visité, ou durant laquelle une infirmière ne peut organiser une visite.
- `soins` <$Patient$> <Elt_1> <Elt_2> ... <Elt_n>

où <*Patient*> est une chaîne (sans espace) représentant le nom du patient, et où <Elt_1> <Elt_2> ... <Elt_n> représente une séquence de soins à administrer successivement à ce patient avec éventuellement quelques contraintes d'espacement entre deux soins consécutifs. La séquence doit commencer forcément commencer par la description d'un soin. Si un élément <Elt_i> correspond à une contrainte d'espacement, alors <Elt_{i-1}> et <Elt_{i+1}> doivent forcément décrire un soin. La contrainte exprime alors lors un écart minimum et/ou maximum de temps, devant s'écouler entre la fin du soin <Elt_{i-1}> et le début du soin <Elt_{i+1}>. Les soins et contraintes d'espacement sont respectivement décrits de la façon suivante :

— un **soin** est décrit par un ou plusieurs codes (séparés par des `+`) correspondant aux actes à administrer au patient durant ce soin (e.g. `INJ`, `INJ+TL1+VCI` ,...) et de façon facultative, par l'indication `:`<*Creneau*> d'un créneau horaire durant lequel ces actes doivent être administrés (le caractère `:` doit suivre sans espace le code du dernier acte).
Le <*Creneau*> peut être décrit par une expression de la forme `[`<Inf>`-`<*Sup*>`]` où <Inf> et <*Sup*> décrivent un horaire conformément au format utilisé pour définir la journée (e.g. `[13h-15h45]`). On peut éventuellement omettre l'un d'entre eux (mais pas les deux). Dans ce cas, on retiendra selon le cas la valeur $InfJ$ (pour la borne inférieure manquante) ou $SupJ$ (pour la borne supérieure manquante) déclarée dans la ligne débutant par `journée`. Pour alléger l'écriture on peut aussi utiliser une constante parmi `matin`, `midi`, `apres-midi` ou `soiree`. Ces constantes peuvent être vues respectivement comme des raccourcis pour les intervalles `[`$InfJ$`-11h]`, `[11h-15h]`, `[15h-19h]` et `[19h-`$SupJ$`]`.

— une **contrainte d'espacement** entre deux soins successifs est de la forme $Inf <, < Sup$ ou $Inf << Sup$, où Inf et Sup suivent la syntaxe des horaires. Elles expriment que l'intervalle de temps s'écoulant entre entre les soins <Elt_{i-1}> et <Elt_{i+1}> doit être respectivement "au minimum de Inf", "au maximum de Sup" ou "au minimum de Inf et au maximum de Sup".

- `adresse` <*Patient*> <*Adresse*>
  où <*Patient*> est un nom de patient et où <*Adresse*> capture toute la fin de la ligne après le nom du patient (qui peut donc contenir des espaces) et correspond à la description de l'adresse du patient (*remarque :* on utilise également cette entrée pour décrire l'adresse du cabinet d'infirmier en utilisant la chaîne `cabinet` à la place du nom d'un patient).
- `acte` <*Code*> <*Duree*> <*Description*>
  où <*Code*> est un code d'acte de soin infirmier, <*Duree*> est un entier décrivant la durée requise pour cet acte (en minutes), et <*Description*> capture toute la fin de la ligne après la durée (elle peut donc contenir des espace) et correspond à une description textuelle de cet acte.
- Toute ligne ne commençant pas par l'un des cas précédents sera assimilée à un commentaire.

Par exemple, le texte ci-dessous ci dessous, décrit quelques lignes extraites d'une description d'instance instance :

```
// Exemple illustrant la syntaxe
journee 7h 20h

adresse cabinet              2 allée des tilleuls, Dourdan
infirmiere Sophie

soins   Jean_DUPONT TL1+MD1:matin          TL2+MD2: soir

soins   Marie_BEL   TSA+MD1:[8h30-10h00]  PS1:[12h30-14h00] MD1+IID:soir
soins   Marie_BEL   CGL:[7h30-8h]

soins   Eric_BRY    IIM       2h<<3h             CSUH          4h<<6h       MD1
indisponibilite Eric_BRY    14h 16h

adresse   Jean_DUPONT        8 Place du Patrouillot, 91530 Le Val-Saint-Germain
adresse   Marie_BEL          19 Sente des Sablons, 91530 Saint-Chéron
adresse   Eric_BRY           7 Rue Henri Dunant, 91410 Dourdan, France

acte          TSA           5           Prise de tension artérielle
acte          CGL          10           Contrôle Glycémie
acte          TL1          15           Toilette légère
```

```
acte          TL2          25           Toilette complète
acte          MD1           5           Administration médicamenteuse orale
acte          MD2          10           Administration médicamenteuse lente
acte          IID              5   Injection intradermique
...
```

On y apprend entre autres choses que la plage horaire pour effectuer toutes les visites ce jour là s'étend de 7h du matin à 20h, et que Sophie est une infirmière (sans contrainte horaire particulière) qu'il y a trois patients nécessitant divers soins :

1. Jean DUPONT nécessite 2 visites, prévues le matin et le soir. La visite du matin comporte 2 actes (une toilette légère et une administration de médicaments) et nécessitera donc 15+5 = 20mn.
2. Marie BEL a 4 visites de prévues, chacune avec un créneau horaire bien défini. On remarque au passage que tous les soins concernant une même personne ne figurent pas forcément sur une même ligne. Cela peut être utile pour distinguer les soins récurrents de ceux plus ponctuels.
3. Eric BRY lui a trois soins de prévus dans la journée, sans contraintes horaires particulières mais devant être espacés au minimum de 2h et au maximum de 3h pour les 2 premiers, et entre 4h et 6h pour le second et le troisième. On apprend de plus qu'il est indisponible pour recevoir des soins de 14h à 16h.

Les données de l'instance renseignent également sur l'adresse du cabinet, celles de chacun des patients et précisent les caractéristiques des actes.

Il est important de comprendre que les patients n'ont pas forcément tous des adresses différentes. Des patients peuvent habiter dans un même immeuble (ou une même maison de retraite). Il est donc nécessaire de bien distinguer les lieux, les patients et les visites.

Le fait de pouvoir répartir les différentes lignes décrivant une instance dans plusieurs fichiers différents doit être vu comme une commodité. Cela permet de réutiliser certaines parties entre plusieurs instances ayant des parties communes. Par exemple la liste des actes répertoriées par Sophie s'est constituée au fil du temps et peut être réutilisée dans les différentes instances (vous pourrez réutiliser le fichier `actes.txt` figurant dans le dossier `data`). Cela peut aussi être précieux pour mettre au point vos jeux de tests, permettant de vérifier que tous les paramètres d'une instance sont bien pris en compte (en créant des variantes proches d'instances).

II) Le travail demandé

L'objectif ultime est d'arriver à générer un planning des visites pour l'ensemble du cabinet, prenant en compte toutes les caractéristiques d'une instance. On se propose d'aborder résoudre le problème progressivement, en prenant en compte successivement différentes caractéristiques de l'instance. A cet effet vous proposerez les modèles suivants :

visites1 Dans un premier temps on fera abstraction des toutes les contraintes horaires sur les différentes visites comme celles d'espacement entre visites ou encore celles des indisponibilités. On cherchera juste à minimiser le temps total de travail de l'infirmière considérée (i.e entre le moment ou elle part de son cabinet avant sa première visite, et celui ou elle termine ses transmissions après la dernière visite).

visites2 Dans ce second modèle devra en plus tenir compte de toutes les contraintes liées aux créneaux horaires, aux contraintes d'espacement entre soins et aux possibles indisponibilités. Une contrainte d'indisponibilité de l'infirmière sera interprétée comme l'impossibilité qu'une quelconque visite puisse avoir lieu (même partiellement) durant l'intervalle de temps spécifié.

visites3 Dans le troisième modèle, on souhaite coordonner le travail de plusieurs infirmières travaillant dans le même cabinet. On s'autorise donc à déclarer plusieurs infirmières (avec ou sans plages horaires associées). Le modèle à réaliser doit permettre de répartir de façon optimale les visites à effectuer entre les différentes infirmières. Cependant, à l'usage Sophie se rend compte qu'un simple cumul des temps de travail des différentes infirmières, donne la plupart du temps

des résultats peu satisfaisants (parfois cela conduit à attribuer toutes les visites à une seule infirmière). On cherchera donc plutôt à minimiser le maximum des temps de travail pour chaque infirmière, de façon à répartir plus efficacement les visites entre les différentes infirmières.

Pour mener à bien ce travail différentes étapes sont à franchir :

1. vous devez d'abord proposer des structures de données adaptées à la représentation des données d'une instance et écrire une fonction, capable de récupérer les données d'une instance à partir du (des) fichier(s) qui la caractérise(nt).
2. Ensuite, il faut concevoir des fonctions permettant de récupérer les coordonnées gps des points correspondant aux différentes adresses de l'instance, ainsi que les temps de parcours entre ces différents points d'intérêt.
3. Ensuite, il vous faudra concevoir les différents modèles demandés. Il est important de bien comprendre à ce stade, ce qui est connu et inconnu.
4. Vous ferez en sorte de pouvoir lancer vos modèles en vous plaçant votre répertoire `src` par un appel de la forme la forme :
   ```
   oplrun visitesX.mod ../instances/instanceY.dat ../data/dat/adresses_essonne_91-410_870_530.dat
   ```
   (le second argument doit correspondre au chemin relatif jusqu'au fichier `.dat` décrivant l'instance. Le troisième argument ne sera nécessaire que si vous utilisez effectivement le fichier d'adresses du cadastre pour rechercher les coordonnées géographiques).
   Chaque fichier d'instance `instanceY.data` contiendra deux informations :
   - `nom` (une string) précisant le nom de l'instance (veillez à ce qu'ils soient tous différents)
   - `fichiersDonnees` (un ensemble de chaines) correspondant aux chemins d'accès vers les fichiers `.txt` caractérisant les données de cette instance (à placer dans le répertoire `data`)
5. Vous ferez en sorte que chaque modèle *visitesX* puisse enregistrer les résultats obtenus sur une instance dont le nom est *instanceY* dans un fichier nommé `InstanceY_visitesX.txt`, que vous enregistrerez dans le répertoire `resultats`.
   Le format exact des fichiers de résultats **vous sera précisé ultérieurement** sur la page du cours mais il devra permettre de décrire, pour chaque infirmière, les patients quelle doit visiter et préciser dans quel ordre avec les horaires de passage escomptés. Il devra aussi mentionner la valeur optimale du critère de décision.
6. Selon la façon dont on s'y prend, le troisième modèle peut sembler plus délicat à mettre en oeuvre. Une suggestion : faites en sorte de rendre explicite dans vos variables de décision, l'infirmière en charge de chaque visite. Et bien qu'elles partent toutes du même endroit (le cabinet), il peut être commode de nommer différemment les points de départ (et donc de fin) correspondant aux trajets de chaque infirmière (même si en pratique ils correspondront toujours à la même adresse). Notez également qu'il n'est pas nécessaire que ce soit toujours la même infirmière qui fasse les différentes visites d'un même patient.

III) Exploitations des sources d'informations géographiques

La construction d'un programme de visites optimal nécessite de pouvoir récupérer les temps de parcours entre les lieux devant être visités. Le projet OMRS offre un certain nombre de services web permettant notamment de construire des itinéraires, de générer des listes d'instructions de déplacements (comme celles d'un gps), etc. Parmi ces services, le service `table` permet notamment, à partir d'une suite de points (représentés par leurs coordonnées GPS), de récupérer les temps de transports entre ces différents points. Les résultats des requêtes sont renvoyés sous la forme d'une chaîne de caractères au format JSON (JavaScript Object Notation).

a) JSON et le langage de script

Le format JSON (`http://www.json.org`) est un format d'échange de données relativement simple, qui permet de représenter différentes sortes de valeurs, pouvant correspondre à :

- des **valeurs atomiques** (des chaînes de caractères, des nombres ou une constante parmi `true`, `false` ou `null`)
- des **séquences** de valeurs (représentées sous la forme `[`$Valeur_1$`, ..., `$Valeur_n$`]`)
- des **objets** que l'on peut assimiler à une collections de paires de types clé/valeurs de valeurs (représentés sous la forme `{`Cle_1 : $Valeur_1$`, ..., `Cle_n : $Valeur_n$`}`, les clés étant elles même des chaînes de caractères).

.

Ces différents types de données peuvent se traduire assez naturellement dans les langages de type EmacsScript (dont *IBM ILOG Script for OPL* est un cas particulier). Les valeurs atomiques sont traduites directement par elles mêmes, les séquences se traduisent en structures de type `Array` et les objets en structures de type `Object` (remarque : les objets de type `Array` sont en fait des cas particuliers d'`Object`, pour lesquels la propriété `length` a été définie et associée à un entier - cf documentation).

Si la plupart des langages JavaScript modernes offrent en standard la possibilité de faire simplement cette conversion, ce n'est pas le cas dans *IBM ILOG Script for OPL*. Mais vous trouverez dans la le répertoire `libs` de l'archive, un fichier `simpleJSONParser.js` qui offre une fonction `parseSimpleJSON(s)` permettant d'effectuer de telles conversions. Une fois effectuée, on peut extraire un élément d'une séquence ou une propriété d'un objet, en utilisant la syntaxe classique du langage de script.

b) Interrogation d'un serveur OSRM

Afin de pouvoir facilement interroger le service OSRM, vous trouverez dans `libs/geoServices.js` une fonction `osrm_table(serie, server)`, qui, à partir d'une chaîne représentant une suite de coordonnées GPS proprement formatée et de l'adresse d'un serveur OSRM, renvoie la réponse à une requête de type `table` sous la forme d'un Objet. Le format de la chaine représentant la série de points considérée doit être une suite de coordonnées GPS séparées par des `;` où chaque point est représenté par deux nombres flottants séparées par une virgule, représentant respectivement la longitude et la latitude de ce point (attention à ne pas se tromper dans l'ordre).

Exemple : `"2.036030,48.5224235;2.121720,48.554347;2.014160,48.523597;2.057541,48.5634484"`

Cette fonction interroge le serveur spécifié en utilisant la fonction `IloOplExec(command)` du langage de script, qui permet de lancer un sous processus Unix. La commande qui est lancée utilise la commande `curl` pour faire l'appel au service web service et redirige la réponse dans fichier temporaire. Celui ci est ensuite lu et reconverti en objet du langage de script, à l'aide de la fonction `parseSimpleJSON()`.

Il est à noter que la réquête au service n'aboutit pas toujours. La cause peut-être due au fait que trop d'utilisateurs interrogent le service à un moment donné, ou encore que vous avez déjà utilisé trop intensément le service (sur `router.project-osrm.org`, qui est avant tout un serveur de démonstration, vous n'êtes pas supposés poser plus d'une requête par seconde, au risque de vous faire blacklister - donc prudence). Nous allons tenter d'installer une version locale d'OSRM, sur les machines du département pour limiter ces restrictions (les indications seront données ultérieurement sur la page du cours). Une autre cause possible d'échec peut être qu'il n'existe pas de route possible entre certaines des coordonnées indiquées, pour le mode de déplacement indiqué.

Lorsqu'une requête a bien abouti, l'objet obtenu doit posséder une propriété `code` dont la valeur doit être `''Ok''`. Dans ce cas l'objet réponse doit également avoir une propriété `durations` dont la valeur est une matrice n*n d'entiers (si votre requête comportait n coordonnées gps). L'élément de coordonnées `[i][j]` ce cette matrice est alors un flottant, indiquant la durée (en secondes) pour se rendre du i-eme point au j-ieme point de la liste de coordonnées fournies en entrée (les entrées étant indexées de 0 à n-1). Dans le cadre de ce devoir, ce niveau de précision n'étant pas pertinent, **on retiendra les valeurs arrondies au plus proche, en minutes**.

c) Récupérer les coordonnées GPS d'une adresse

Utiliser le service d'OSRM suppose donc d'avoir préalablement récupéré les coordonnées gps des points d'intérêt. Pour cela, différentes sources d'informations sont possibles.

Utilisation d'informations venant du fichier du cadastre :

Dans le cadre d'une démarche générale d'ouverture des données publiques, le *service du cadastre* diffuse des informations qui précise l'emplacement de la moindre parcelle cadastrale en France. Un extrait des adresses correspondant à des maisons ou immeubles, sur les communes de l'Essonne dont le code postal est `91410`, `91530` ou `91870` a été réalisé et converti directement sous la forme d'une ensemble de tuples de la forme `<Longitude, Latitude, Localisation, CodePostal, Ville, Pays>`. Cet ensemble répertorie 8689 adresses qui peuvent servir de base pour construire des jeux de tests et des instances. Le fichier `adresses_essonne_91-410_870_530.dat` se trouve dans le répertoire `data/dat`.

Attention toutefois à la façon dont vous écrivez une adresse dans vos fichier d'instance. Vous n'écrirez pas forcément l'adresse exactement de la même façon et de façon aussi complète que ce qui est décrit dans le fichier. Comparer une adresse aux éléments figurant dans cette base du cadastre, peut nécessiter quelque étapes de standardisation des adresses, ne serait-ce que pour résoudre des problèmes comme des espaces en trop, des noms en majuscules ou minuscules etc...

Utilisation du service Nominatim :

Si l'adresse recherchée n'a pas pu être localisée dans la base du cadastre, on peut tenter d'utiliser un autre service web comme **Nominatim** (`http://nominatim.org`). Ce service, permet, à partir d'une adresse sous forme de chaine de caractères de récupérer la ou les coordonnées des points connus dans OSM qui semblent les plus proches de l'adresse spécifiée. Il est possible de vous inspirer de la fonction `osrm_table(...)` pour proposer une méthode similaire permettant d'interroger un serveur Nominatim par une requête de la forme : `https://nominatim.openstreetmap.org/search/<query>?<params>`. Pour la syntaxe précise des requêtes, se référer à la documentation (`http://nominatim.org/release-docs/latest/api/Search/`). Pensez à préciser dans les options que vous voulez le résultat au format JSON, de façon a pouvoir ensuite utiliser la fonction de conversion mise à votre disposition. Attention cependant à respecter les délais d'une seconde entre deux requêtes successives. Se construire un cache local peut aussi être un moyen de limiter les requêtes successives.

Remarque sur la précision des coordonnés des points d'OpenStreetMap

Si les informations publiées par le cadastre sont progressivement intégrées dans la base d'OSM, ce travail est toujours en cours et repose sur le bénévolat de tous les volontaires qui participent à l'enrichissement de la base (chacun peut contribuer au projet...). Or sur les communes considérées on peut constater que c'est encore loin d'être achevé.

Lorsque la base d'OSM ne trouve pas de maison ou d'immeuble correspondant aux coordonnées indiquées, le service OSRM prend le parti d'utiliser les coordonnées de l'objet le plus proche qui semble correspondre dans la base OSM. Si vous inspectez plus en détail les résultats de la chaine JSON renvoyé par OSRM, vous pourrez constater que pour chaque coordonnée soumise dans votre requête, il vous renvoie les coordonnées du point qu'il a retenu, et la distance (en mètres) par rapport à au point initialement mentionné.. Si le travail de localisation des différents numéros d'une rue n'a pas été fait, OSRM renvoie généralement les coordonnées de l'objet qui modélise la rue toute entière. Dans ce cas, différentes adresses dans une même rue seront toutes identifiées aux mêmes coordonnées gps.

Dans le cadre de ce projet, si l'adresse précise n'est pas précisément connue par OSRM, nous prendrons le parti de retenir les temps de parcours par rapport aux points retenus par OSRM.

IV) Quelques conseils pour une bonne réalisation du devoir

Vous aurez besoin d'effectuer des opérations d'entrées sorties sur des fichiers (pour lire les données des instances et écrire les résultats). Vous trouverez, les informations utiles sur la façon de procéder, dans la partie de la documentation portant sur *IBM Ilog Script Reference Manual* et plus particulièrement dans la rubrique *OPL Classes* et notamment au sujet des classes `IloOplInputFile` et `IloOplOutputFile`. De façon générale, il faudra utiliser des primitives du langage *IBM Ilog Script* pour ouvrir et lire le fichier ligne a ligne, et en extraire les informations utiles pour alimenter des structures de données *OPL* modélisant les informations brutes de l'instances. Ensuite, vous pourrez faire les prétraitement adéquats pour alimenter vos modèles.

On rappelle que par rapport aux possibilités offertes par le lange *Opl* **il existe des limitations** sur ce que l'on peut faire dans *IBM Ilog Script for Opl*. Notamment, il n'est possible de construire

que des tuples dont les attributs correspondent à des valeurs **atomiques** (donc, ni des ensembles, ni des tableaux). Ceci empêche donc de saisir directement une donnée dans une structure dont l'un des attributs serait de ce type. Il peut donc être nécessaire de passer par certaines structures de données auxiliaires, utiles juste pour enregistrer de façon "à plat" les informations lues dans les fichiers de l'instance, puis de faire ensuite quelques pré-traitements, en OPL pour reconstruire d'autres structures plus élaborées, mieux adaptées à résolution de votre problème.

On rappelle également qu'il est possible dans le langage *IBM Ilog Script for Opl*, d'utiliser des fonctions du langage *Opl*, en préfixant l'appel de la fonction par `Opl.` (exemple `Opl.f(...)`), sous réserve de respecter les types attendus dans Opl.

Pour gagner du temps au moment du test de vos modèles, il n'est pas difficile d'écrire une petite fonction qui, pour un ensemble de modèles et un ensemble d'instances, automatise la résolution de chaque modèle sur chaque instance (comme dans le td 11).

Quelques remarques / recommandations :

- Sur la page du cours : `https://www.lri.fr/~chatalic/Enseignement/ia-lc-m1-app/` vous trouverez prochainement une archive structurée au format demandé. Cette archive contient les librairies et données mentionnées et une instance répondant au format demandé. Vous pouvez l'utiliser pour tester vos fonctions de lecture d'instances. Des instances plus conséquentes seront ajoutées ultérieurement, mais ne les attendez pas pour commencer à travailler.
- Si certains points s'avèrent ambigus et nécessitent des précisions particulières, dans un soucis d'équité, les réponses à vos éventuelles questions seront faites sur la page du cours.
- Par rapport à ce que ce que vous avez déjà pu réaliser jusqu'à présent, la principale différence est qu'il faut ici commencer par lire les données du problème, en extraire les informations pertinentes, récupérer les informations requises. Mais il s'agit là essentiellement de programmation *classique*, indépendant de l'aspect *contraintes*. Ensuite il vous faudra choisir comment prétraiter ces données pour pouvoir alimenter vos modèles.
- La partie concernant la lecture des données peut-être mutualisée par les différents modèles.
- Il faudra vous assurer pour chaque modèle proposé, que chaque propriété à satisfaire est modélisée de façon adéquate. Afin de s'en convaincre, il est fortement conseillé de se créer différentes instances très simples (i.e. avec très peu de données) pour tester séparément les différents aspects du problème (mettez les dans `instances/test`)
- Lorsque votre code vous semblera abouti, vous pourrez concevoir des instances plus complexes, qui mélangent les différents aspects du problème et tester le passage à l'échelle.
- Naturellement, les bons principes de génie logiciel s'appliquent quels que soient les langages. La modularité est toujours un élément essentiel de la qualité du code produit. Elle facilite non seulement la lisibilité mais permet également d'avoir les idées plus claires lors de la conception et facilite le déboggage. Donc n'hésitez pas ici à décomposer votre code OPL Script en un ensemble de petites fonctions indépendantes, qui font des choses précises et limitées (par exemple, pour traiter chaque type de ligne possible dans les fichiers de données).
- Pour ceux qui n'ont pas installé la suite d'IBM sur leur propre machine (ce qui n'est pas vraiment indispensable) il vous est rappelé que vous pouvez vous connecter à distance via ssh sur les machines `tp-ssh1.dep-informatique.u-psud.fr` ou `tp-ssh2.dep-informatique.u-psud.fr` et lancer `oplrun` en mode console. Il est donc tout à fait possible de travailler à distance.
- Vous devez mettre à jour régulièrement votre dépôt git, chaque fois que vous rajouté une nouvelle fonctionnalité.
- Enfin, **n'attendez pas le dernier moment pour vous y mettre !** Votre objectif doit être d'arriver à lire les données des fichiers d'instance dans les meilleurs délais. Cela peut se faire progressivement. Commencez par voir comment récupérer certaines lignes seulement, en ignorant les autres. Puis enrichissez votre code jusqu'à arriver à lire correctement les informations de toutes les lignes du format décrit.
- Soignez la rédaction de votre petit rapport de synthèse sur le fond, en décrivant bien vos modèles et la façon de traduire les contraintes, et commentez vos résultats sur les instances traitées