Mini projet ProLog
Calcul d'un itinéraire au sein d'un réseau de transport en commun

1 Introduction

L'objectif de ce mini-projet est de développer un programme en ProLog permettant de calculer un itinéraire à partir d'un réseau de transport en commun (par exemple la RATP).

Ce projet sera programmé en `swi-prolog` et s'effectuera en **binôme** uniquement. Vous devrez rendre par email [1] et **avant le 31 décembre 2018 à 23h59** [2] une archive portant le nom de votre binôme et contenant les fichiers suivants :

- Un rapport (au format `pdf`) de moins de trois pages recto-verso, expliquant le fonctionnement global de votre programme, les choix que vous avez faits, ainsi que des captures d'écrans montrant un exemple d'utilisation de **chaque** prédicat que vous aurez défini ;
- Un fichier `.pl` **documenté** contenant le code source de votre programme ;
- Un fichier `ReadMe` expliquant les commandes à taper pour exécuter votre programme.

2 Représentation du réseau de transport

Un ligne de transport sera définie par le prédicat `ligne` :

```
ligne(Nom, Type, LArret, Dir1, Dir2)
```

où :
- `Nom` décrit le nom de la ligne
- `Type` décrit le type de moyen de transport (métro, RER [3], tramway...)
- `LArret` est une liste comprenant tous les arrêts de la ligne
- `Dir1` et `Dir2` décrivent les deux directions de la ligne.

Par exemple, on représentera la ligne 1 du métro parisien comme celà :

```
ligne(m1, metro,  [la_defense, esplanade_defense,
                   pont_de_neuilly, les_sablons,
                   porte_maillot, argentine,
                   charles_de_gaulle_etoile,
                   george_V, franklin_d_roosevelt,
                   champs_elysees_clemenceau,
                   concorde, tuileries,
                   palais_royal_musee_du_louvre,
                   louvre_rivoli, chatelet,
                   hotel_de_ville, saint_paul,
                   bastille, gare_de_lyon,
                   reuilly_diderot, nation,
                   porte_de_vincennes, saint_mande,
                   berault, chateau_de_vincennes],   la_defense, chateau_de_vincennes).
```

1. `elise.bonzon@parisdescartes.fr` et `julien.rossit@parisdescartes.fr`
2. Et donc, les projets reçus en 2018 seront notés et corrigés, ceux reçus en 2019 ne seront même pas ouverts. Aucune excuse ne sera acceptée.
3. Par souci de simplification, on ne considère ici que les arrêts de RER se trouvant dans Paris.

Un fichier contenant les lignes du RER, du métro et du tramway parisien est disponible sur la page Moodle du cours.

Afin de pouvoir travailler plus aisément sur le réseau donné, on se pose le problème de rajouter les prédicats suivants dans la base de connaissance.

1. Ecrivez un prédicat

   `create_nbstations.`

   qui ajoute un prédicat `nb_stations(NomLigne, NbArrets)` qui associe à chaque ligne `NomLigne` du réseau de transport le nombre de stations présentes sur cette ligne [4].
   Par exemple, l'appel de `create_nbstations.` ajoutera *entre autres* le prédicat `nb_stations(m1, 25)` dans la base de connaissances.
2. Ecrivez un prédicat

   `create_numstation.`

   qui ajoute un prédicat `num_stations(NomStation, NomLigne, Dir1, NumD1, Dir2, NumD2)` qui associe à chaque station `NomStation` de la ligne `NomLigne` le numéro de la station dans chacune des deux directions de cette ligne [5].
   Par exemple, l'appel de `create_numstation.` ajoutera *entre autres* le prédicat `num_stations(chatelet, m1, chateau_de_vincennes, 15, la_defense, 11)` dans la base de connaissances [6].

3 Calcul d'itinéraires

On considère à présent le problème de savoir si une ligne passe par deux arrêts donnés, ainsi que le problème qui consiste à compter le nombre de stations séparant ces deux arrêts le cas échéant.

3. Ecrivez un prédicat

   `station(Station, LLignes)`

   qui retourne vrai si la liste `LLignes` contient toutes les lignes passant par la station `Station`. [7]
4. Ecrivez un prédicat

   `intersection(Ligne1, Ligne2, LStations)`

   qui retourne vrai si la liste `LStations` contient toutes les stations dans lesquelles les lignes `Ligne1` et `Ligne2` se croisent.
5. Ecrivez un prédicat

   `correspondance(Ligne, LLignesStations)`

   qui retourne vrai si la liste `LignesLStations` contient toutes les correspondances possibles sur la ligne `Ligne`. `LignesLStations` devra donc contenir toutes les paires `[nomligne, station]` qui permettent d'effectuer une correspondance sur la ligne `Ligne`.
6. Ecrivez un prédicat

   `dessert(Ligne, Depart, Arrivee)`

   qui retourne vrai si la ligne `Ligne` dessert les arrêts `Depart` et `Arrivee`.
7. Ecrivez un prédicat

   `nbarret(Ligne, Depart, Arrivee, Dir, NbArrets)`

   qui calcule dans la variable `NbArrets` le nombre d'arrêts existants entre les arrêts `Depart` et `Arrivee` sur la ligne `Ligne` si on la prend dans la direction `Dir` [8].

4. Le prédicat `assert` pourra vous être utile.
5. Le prédicat `forall` pourra vous être utile.
6. Puisque sur la ligne 1 du métro, la station Chatelet est la 15ème depuis La Défense si on prend la direction Château de Vincennes, et la 11ème depuis Château de Vincennes si on prend la direction La Défense.
7. L'étude des prédicats prédéfinis `setof`, `bagof` et `findall` peut s'avérer utile pour celà.
8. Attention, si la direction n'est pas la plus naturelle, il faut faire un tour complet de la ligne.

On considère à présent le problème qui consiste à savoir s'il existe un itinéraire entre deux arrêts (éventuellement avec des changements de moyen de transport). *Avant de programmer le prédicat suivant, lisez attentivement la suite du sujet pour savoir quels sont les informations nécessaires dans votre variable* `Trajet`.

8. Ecrivez un prédicat

   `itineraire(Depart, Arrivee, Trajet)`

   qui est vrai quand `Trajet` décrit un itinéraire permettant de partir de l'arrêt `Depart` pour arriver à l'arrêt `Arrivee`.

9. Ecrivez un prédicat

   `listeitineraire(Depart, Arrivee, LTrajet)`

   qui est vrai quand `LTrajet` est une liste qui contient *tous* les itinéraires permettant de partir de l'arrêt `Depart` pour arriver à l'arrêt `Arrivee`.

10. Ecrivez un prédicat

    `tempstrajet(Trajet, Tps)`

    où `Tps` est le temps estimé (en *minutes*) du trajet `LTrajet` si l'on suppose que le temps de trajet entre deux stations est d'une minute, et qu'une correspondance nécessite 5 minutes.

11. Ecrivez un prédicat

    `reseau_emprunte(Trajet, LReseau)`

    qui est vrai quand `LReseau` est une liste qui contient les noms de *tous* les réseaux (métro, RER, tramway) empruntés lors du trajet `Trajet`.

12. Ecrivez un prédicat

    `nb_stations(Trajet, NbStations)`

    où `NbStations` est le nombre de stations traversées au cours du trajet `Trajet`.

13. Ecrivez un prédicat

    `nb_correspondances(Trajet, NbCorres)`

    où `NbCorres` est le nombre de correspondances effectuées au cours du trajet `Trajet`.

4 Prise en compte des préférences d'un utilisateur

Ecrire à présent un prédicat [9] permettant de trouver un itinéraire en tenant compte des préférences de l'utilisateur sur le trajet. Les choix disponibles pour spécifier les préférences des utilisateurs sont les suivants :

- choix du réseau (métro, RER, tramway) préféré [10] ;
- choix du trajet comprenant le moins de stations traversées ;
- choix du trajet ayant le moins de correspondance ;
- choix du trajet au cours duquel l'utilisateur emprunte le moins de réseau de transport différent ;
- choix du trajet le plus court (en minutes).

5 Question subsidiaire [11]

Ecrivez un prédicat permettant de décider s'il existe un *chemin hamiltonien* dans le plan de métro parisien. On rappelle qu'un chemin hamiltonien est défini comme étant un trajet passant par toutes les lignes une et une seule fois. Il n'est donc pas nécessaire de passer par toutes les stations. En outre, tentez d'interdire les correspondances multiples à une seule station.

9. Définir plusieurs prédicats intérmédiaires sera peut-être nécessaire pour cela...
10. On suppose ici que l'utilisateur cherche à n'emprunter *que* son réseau préféré (et que donc s'il n'existe pas d'itinéraire complet sur le réseau choisi, le prédicat retournera *faux*.
11. Mais néanmoins importante...