Rapport sur l'Implémentation d'itinéraires en Prolog

Le projet vise à assister un utilisateur d'un réseau de transport à trouver un itinéraire entre deux stations. ce dernier doit prendre en compte diverses conditions telles que les horaires de départ et d'arrivée et vise a minimiser la durée du voyage.

La creation d'un programme en Prolog qui permet a l'utilisateur de simplement entrer le premier arret ou il veux commencer son trajet, et un deusieme arret ou il descendras, et le progeam prondras soins de proposer un trajet optimal.

Pour les predicats escentiel on a :

addh/3

Ce predicat permet d'ajouter deux heures, qui sont sous forme [Heure, Minute]. Il commence par ajouter les minutes ensemble puis devise le resultat par 60, puis il prend le resultat de la division et l'ajoute au heure. Apres, il remplace les minute par le modulo des minutes par 60 pour assurer qu'il n'y est pas une valeur superieur a 60.

affiche/1

Ce predicat a un seul parametre sous forme [Heure, Minute]. Il commance par prendre l'element "Heure" puis ajoute un h apres, ensuite il ajoute l'element "Minute" .

lig/3

Ce predicat prend trois parametre , les deux premier parametres represente deux arret "Arret1" et "Arret2", et le troisieme parametres et une ligne. Ce predicat permet de verfier si la ligne donner passer pas les deux arrets, il fait appele au predicat getArrets/2 pour avoir tout les arrets d'une ligne, puis fait appel au predicat membre/3 pour verifier d'abord si la ligne passe par "Arret1" ensuite verifie si "Arret2" est dans arrets apres "Arret1".

getArrets/2

Ce predicat permet de retourner tout les arrets d'une ligne, il prend un parametre en entrer et en retourne un, le premier c'est la ligne dont on veut extraire les arrets, et le deuxieme c'est les arrets retourner. Il fait appel au predicat ligne(Ligne,_,X,_,_), il sauvegarde les arrets de Ligne dans la variable "X" et ignore le reste (en utilisant "_").

getDep/4

Ce predicat permet de retourner les information du trajet d'aller arrets d'une ligne, il prend un parametre en entrer et en retourne un,le premier c'est la ligne dont on veut extraire les information du trajet, et le deuxieme une liste pour l'heure et les minutes du debut, le troisieme c'est le nombre de minute utiliser comme interval des departs et le quatriem c'est une liste pour l'heure et les minutes de la fin du trajet.

getDeps/4

Ce predicat fait appel a lig/3 puis getDep/4, il permet de retourner les information du trajet d'aller arrets d'une ligne sachant que cette ligne passe par deux arret specifique.

membre/3

Ce predicat prend en entrer deux parametres (un arret et une liste d'arrets) et en retourne un, il verifie si l'arret appartient a la liste des arrets, si il le trouve il retourne la liste des arretes qui sont apres lui. Il fait cela en verifiant si l'arret est la tete de notre liste, sinon il fait un appel recursive pour verifier si il est la tete des elements de la queue ansi de suite.

convertMin/2

Ce predicat prend en parametres une heure a convertir et une variable pour le resultat, il permet de convertir une heure en minute, il multiplie le nombre des heures par 60 et les ajoutes au minutes.

ligtot/4

Ce predicat prend quatre parametres, il est vrai quand Ligne part le plus tôt possible après Horaire parmi les lignes qui vont de l'Arret1 à l'Arret2. cree une liste de tout les ligne qui passes de l'Arret1 à l'Arret2 et les met dans une liste de liste qui contient des informations sur l'heure du depart, le temps que la Ligne attend avant qu'il y aura un autre départ, puis fait appel a la fonction closestDepart qui retourn la ligne voulu.

closestDepart/6

Ce predicat est un predicat recursive, il prend en parametre une liste de ligne, un horaire. il convertis les heures en minutes pour simplifier le travail, puis il verifie si la valeur du depart de la ligne est inferieur a la l'heure donnes, il incremente ca valeur par le temps que la Ligne attend avant qu'il y aura un autre départ, jusqu'a quand elle est superrieur, sinon la distance est la distance actuel. puis il compart la valeur donner dans l'appel recursive, et fait un appel recursive avec la plus petit valeur. La condition d'arret est quand on a terminier de parcourir le tableau.

ligtard/4

Ce predicat prend quatre parametres, il est vrai quand Ligne part le plus tard possible avant Horaire parmi les lignes qui vont de l'Arret1 à l'Arret2. cree une liste de tout les ligne qui passes de l'Arret1 à l'Arret2 et les met dans une liste de liste qui contient des informations sur l'heure du dernier depart, le temps que la Ligne attend avant qu'il y aura un autre départ, puis fait appel a la fonction closestDepart qui retourn la ligne voulu.

closestArrive/6

Ce predicat est un predicat recursive, il prend en parametre une liste de ligne, un horaire. il convertis les heures en minutes pour simplifier le travail, ensuite il calcule l'heure de l'arriver (l'heure du dernier

depart plus le temps que la ligne met pour partir du premier au dernier arret),puis il verifie si la valeur de l'arriver de la ligne est superriere a la l'heure donnes, il decremente ca valeur par le temps que la Ligne attend avant qu'il y aura un autre départ, jusqu'a quand elle est inferieure, sinon la distance est la distance actuel. puis il compart la valeur donner dans l'appel recursive, et fait un appel recursive avec la plus petit valeur. La condition d'arret est quand on a terminier de parcourir le tableau.

Exemples d'Utilisation

?- ligtot(jaures,stalingrad,L,[4,0]).

L = 2.

La ligne qui part en premier apres 4h de jaures a stalingrad est la ligne 2

?- ligtard(jaures,stalingrad,L,[2,46]).

L = 2.

La ligne qui arrive en dernier avant 2h46 de jaures a stalingrad est la ligne 2

Conclusion

Malheuresement pour par soucis de clarté et de temps on a pas pu cree un projet complet et sans défaillance, cependant on a fait de notre mieux pour essayer de resoudre les problems proposer.