Tests

Banditcpp est un framework qui vise a améliorer notre expérience lors de la création de test unitaires pour notre application.
Ce framework est disponible gratuitement sur GitHub.
Pour le faire fonctionner au sein du projet, il faut placer le dossier bandit a la racine de celui ci.
Il faut aussi savoir qu'il est dépendant d'un autre framework : « snow …. » (trou de memoire).

L'utilisation de ce Framework nous a été conseillé par notre client.
Bien que nous ne connaissions pas son fonctionnement, celui-ci est simple a comprendre est facile d'utilisation.

Classe stationfilter

Pour tester la classe, on peut commencer par définir son comportement au sein de l'application.
A son instanciation (lors de sa construction), elle recoit en parametre StationFilterParams, qui permet de régler les bornes.
La fonction « filter » reçoit un « Qvector » de stations pour lesquelles elle filtre l'attribut « tripsflow » en fonction des paramètres que nous avons préalablement réglé. Elle renvoie ensuite un QVector des stations apres l'appliquage du filtre.
Ses paramètres nous permettent de régler un minimum et un maximum.
Pour tester cette fonction, nous avons testé si les bornes étaient bien réglés. Pour cela nous avons crée des stations, pour lesquelles nous initialisons la variable « tripsflow ».

TRIPS FLOW MIN :

Pour tester la borne minimum de cette fonction, on initialise la variable minTripsFlow de StationFilterParams a 51.
Nous initialisons deux stations avec une variable tripsflow a 51 (pour la stations 1) et l'autre a 50 (station 2).
Pour la réussite du test, il faut que la station 1 soit acceptée (51 est supérieur ou égal a 51 …) tandis que la station 2 doit être refusée (50 inférieur a 51).

(test validé)

TRIPS FLOW MAX :

Le second test suit la même démarche appliquée au maximum. On initialise maxTripsFlow a 90, puis on initialise les valeurs de tripsflow pour deux stations (station 3 : 90, station 4 : 91).
Pour la réussite du test, il faut que la station 3 soit acceptée (90 inférieur ou égale a 90) mais pas la station 4 (91 supérieur a 90)

(test validé)

TRIPS FLOW INVERT MIN AND MAX :

Le troisième test consiste a observer le comportement de la fonction lorsqu'on inverse les paramètres. Nous initialisons donc minTripsFlow a 90 (valeur supérieur) et maxTripsFlow a 60 (valeur inférieur). Nous testons cette nouvelle disposition sur les instances de classe 3 et 4.
Pour la réussite du test, aucune station ne doit être renvoyé par ce filtre.

(test validé)

tripSelector

HOUR MIN :

Cette classe, a sa construction, prend en paramètre une instance de la classe TripsSelectionParams, qui permet de régler les bornes.
Sa fonction selectfrom recoit un Qvector de « Trip » pour lequelle elle filtre la periode (en heure). Elle vérifie aussi si elle connaît bien une station parmis les stations de départ ou d'arrivée des trips. et elle renvoie un Vector » des trips filtrées.
Pour tester cette fonction, nous allons créer plusieurs trip pour lesquelles nous allons initialiser quatres paramètres :

- startDateTime : Date de départ (En heure)
- endDateTime : Date d'arrivée (En heure)
- startStationId : Id de la station de départ
- endStationId : Id de la station d'arrivée

Nous allons la aussi vérifier le bon fonctionnement des bornes en testant le minimum et le maximum, l'inversion des deux, et la validation de la station.

Pour le premier test, nous allons tester la borne minimale en créant deux trips différents. Nous modifions donc les paramètres cité au dessus. Voici l'initialisation des deux classes t1 (trip 1) et t2 (trip 2).
t1 :

startdatetime : 05 / 05 / 2016 ; 13h50
enddatetime : 05 / 05 / 2016 ; 14h30
startstationid : 1
endstationid: 1

t2 :

startdatetime : 05 / 05 / 2016 ; 14h00
enddatetime : 05 / 05 / 2016 ; 14h01
startstationid : 1
endstationid: 1

nous initialisons la fonction avec params.fromhour = 14 et params.tohour = 15. Ainsi, le premier trajet, commencant a 13h50, ne doit pas etre accepté par le filtre, tandis que le deuxième doit l'être.

(test validé)

HOUR MAX :

En partant du même principe, nous allons tester la borne maximale.
t3 :

startdatetime : 21 / 04 / 2016 ; 15h40
enddatetime : 21 / 04 / 2016 ; 16h00
startstationid : 1
endstationid: 1

t4 :

startdatetime : 21 / 04 / 2016 ; 15h40
enddatetime : 21 / 04 / 2016 ; 16h50
startstationid : 1
endstationid: 1

nous initialisons la fonction avec params.fromhour = 15 et params.tohour = 16. Ainsi, le premier trajet, finissant a 16h00, doit etre accepté par le filtre, tandis que le deuxième de doit pas l'être (16h50 > 16h00).

Dans un premier temps, ce test a échoué ; en effet nous comparions seulement les heures (sans prendre en compte les minutes) ce qui ne gérait pas ce type de cas. *L'application a été mise a jour pour corriger ce problème.* (a vérifier)

STATION VERIFICATION :

classe tripsfilter :

Cette classe vise a filtrer les trips. Lors de sa construction, elle prend en paramètre une instance de la classe TripsFilterParams, qui permet de modifier plusieurs paramètres pour gérer les filtres :

- minDistance : distance minimale du trajet
- maxDistance : distance maximale du trajet
- minDurration : durée minimale du trajet
- maxDuration : durée maximale du trajet
- fromPeriod : date de départ
- toPeriod : date d'arrivée

La fonction « filter » vise a appliquer les filtres a un ensemble de trajets (Qvector de trip). Elle renvoit un ensemble de trajet filtrés.

DISTANCE MIN

Nous créons deux trajets t1 et t2 pour lesquels on définit les paramètres suivant : (le paramètre important est distance, le reste sera compris dans les filtres)

t1 :

- id : 1
- distance : 19
- duration : 400
- startDateTime : 07/07/2016
- endDateTime : 07/07/2016

t2 :

- id : 2
- distance : 20
- duration : 400
- startDateTime : 07/07/2016
- endDateTime : 07/07/2016

On définit ensuite les paramètres du filtre, le but est de tester le paramètre minDistance.

- minDitance : 20
- maxDistance : 100
- minDuration : 0
- maxDuration : 20000
- fromPeriod : 01/01/2016
- toPeriod : 30/12/2016

Pour la réussite du test, il faut que le trajet 1 (t1) soit refusé, tandis que t2 doit etre accepté.

(test validé)

<u>DISTANCE MAX</u>

t1 :

- id : 1
- distance : 180
- duration : 400
- startDateTime : 07/07/2016
- endDateTime : 07/07/2016

t2 :

- id : 2
- distance : 181
- duration : 400
- startDateTime : 07/07/2016
- endDateTime : 07/07/2016

On définit ensuite les paramètres du filtre, le but est de tester le paramètre maxDistance.

- minDitance : 20
- maxDistance : 180
- minDuration : 0
- maxDuration : 20000
- fromPeriod : 01/01/2016
- toPeriod : 30/12/2016

Le trajet t3 doit etre accepté tandis que t4 ne doit pas l'être.

(test validé)

INVERT DISTANCE MIN AND MAX

On garde les meme trajets et le meme filtre, la seule différence est que l'on inverse les valeurs du maximum et du minimum :

- minDitance : 20
- maxDistance : 180

Il faut qu'aucun trajet ne soit accepté.

(test validé)

DURATION MIN

Classe station sorter

Cette classe permet de trier dans l'ordre décroissant les stations données en fonction de plusieurs éléments.
Voici les différents forme de tri possible :

- arrivals : nombre de trajets arrivants
- cycles : nombre de stations cycliques
- departures : nombre de trajets sortants
- distance : distance moyenne des trajets
- duration : durée moyenne des trajets

Pour les tester, nous allons créer 3 stations différentes pour lesquelles nous allons

ARRIVALS

On initialise plusieurs stations (s1, …, sn) :
On crée plusieurs trajets pour les mettre en trajets « arrivants » (3 pour s1, 5 pour s2 et 7 pour s3).
Le résultat a obtenir est, dans l'ordre : s3, s2, s1 (car s3 > s2 > s1).

(test validé)

CYCLES :

Pour les mêmes stations, nous créons des trajets que nous allons définir en temps que trajets « cycliques » (5 pour s1, 7 pour s2, 2 pour s3).
Le résultat a obtenir est, dans l'ordre : s2, s1, s3 (car s2 > s1 > s3).

(test validé)

DEPARTURES :

Même démarche, nous allons crée des trajets que nous allons définir en trajets « sortants » (7 pour s1, 5 pour s2 et 2 pour s3).
Le résultat a obtenir est, dans l'ordre : s1, s2, s3 (car s1 > s2 > s3).

DISTANCE :

Nous avons juste a modifier la valeur de la variable avgTripDistance (10 pour s1, 30 pour s2 et 0 pour s3).
Le résultat a obtenir est, dans l'ordre : s2, s1, s3 (car s2 > s1 > s3).

DURATION :

Nous avons modifié la valeur de la variable avgTripDuration (10 pour s1, 30 pour s2 et 50 pour s3).
Le résultat a obtenir est, dans l'ordre : s3, s2, s1 (car s3 > s2 > s1).

(DOIT ON TESTER LES SET?)