Compte-rendu du TP C++ 2
Tests

Marc Gagné Selma Nemmaoui

Novembre 2015

Préambule

Afin de vérifier la qualité de notre code, nous avons mis en place une procédure de non-régression dans le répertoire `/tests/`. Elle comprend trois parties :

— des fichiers texte `.in` et `.out` qui fournissent des entrées pour le programme ainsi que la sortie attendue - si la sortie obtenue correspond au contenu du fichier `.out`, le test est validé (ces tests sont détaillés par la suite),

— un script Python `tp-oo_2-gen.py` qui permet de générer des fichiers `.in` et `.out` de grande taille, ce qui permet de vérifier que notre programme peut gérer de grandes quantités de données,

— et un script bash `tp-oo_2-test.sh` qui vérifie automatiquement plusieurs tests avec une seule commande ; le chemin de l'exécutable doit lui être passé en paramètre.

Le script Python peut être utilisé pour générer une infinité de tests, soit en mode normal où chaque capteur, chaque minute a un événement (utile pour tester la commande `OPT`), ou en mode `--random` (pour tester les autres commandes). Le script bash permet de tester tous les tests décrits ci-dessous ; si le fichier d'entrée pour le test MAX.6 n'est pas trouvé, le test n'est pas exécuté.

Note : la commande `EXIT` n'a pas de test associé, car son fonctionnement est extrêmement simple et elle est utilisée dans chaque test.

ADD.1

Fichiers : `add.1.in`, `add.1.out`

Le but de ce test est de vérifier que la commande `ADD` fonctionne correctement. Elle suppose que `STATS_C` fonctionne correctement pour des valeurs ordinaires (le test suivant vérifie plus strictement que cette commande fonctionne bien).

— Le test vérifie que chaque fois que la commande est appelée, le bon événement est rajouté. Pour cela, il vérifie d'abord qu'il n'y a aucun événement associé au capteur 42 en vérifiant que la commande `STATS_C` ne renvoie rien.

— Ensuite, le test ajoute un événement au capteur 42, puis vérifie que la commande `STATS_C` reflète l'ajout de cet événement.

— Le test essaie ensuite de voir si l'ajout d'un événement au capteur 41 ne modifie pas les statistiques du capteur 42.

— Finalement, un autre événement est ajouté au capteur 42, et le test vérifie si les statistiques de ce capteur ont bien été modifiées.

Pour tous les tests subséquents, il est supposé que cette commande fonctionne correctement.

STATS_C.2

Fichiers : `stats_c.2.in`, `stats_c.2.out`

Le but de ce test est de vérifier que la commande `STATS_C` fonctionne correctement.

— Le test vérifie d'abord que la commande ne renvoie rien sans ajout d'événements.

— Plusieurs événements d'un même capteur sont ensuite ajoutés, et le test vérifie que les statistiques affichées reflètent bien la répartition des états.
— D'autres événements d'un autre capteur sont ajoutés, et le test vérifie que les statistiques de chaque capteur sont indépendantes l'une de l'autre.
— Finalement, d'autres statistiques sont rajoutées au capteur initial et le test vérifie que les statistiques sont modifiées correctement.

JAM_DH.3

Fichiers : `jam_dh.3.in`, `jam_dh.3.out`

Le but de ce test est de vérifier que la commande `JAM_DH` fonctionne correctement. Pour ce faire, il ajoute 4 événements à plusieurs capteurs le même jour de la semaine, vérifiant que les statistiques d'embouteillage reflètent chaque ajout correctement, puis il ajoute un événement d'un autre jour de la semaine pour s'assurer que les statistiques pour le jour initial ne sont pas perturbées.

STATS_D7.4

Fichiers : `stats_d7.4.in`, `stats_d7.4.out`

Le but de ce test est de vérifier que la commande `STATS_D7` fonctionne correctement. Le test marche comme le test JAM_DH.3 : il ajoute sept événements sur le même jour de la semaine, vérifie que les statistiques de trafic sont correctes, puis ajoute sept autres événements à un autre jour de la semaine et vérifie que les statistiques de chaque jour sont indépendantes l'une de l'autre.

OPT.5

Fichiers : `opt.5.in`, `opt.5.out`

Le but de ce test est de vérifier que la commande `OPT` fonctionne correctement.
— Il ajoute d'abord beaucoup d'événements pour trois capteurs le même jour de la semaine, la majorité des événements ayant lieu directement l'un après l'autre pour assurer la continuité temporelle.
— Un premier essai de la commande `OPT` vérifie qu'il est impossible de trouver un chemin valable avant la séquence continue d'événements.
— Les deux essais subséquents essaient de trouver deux chemin valables sur deux parties de la plage horaire continue.
— Le dernier essai vérifie qu'il est impossible de trouver un chemin après la séquence continue d'événements.

MAX.6

Fichiers : `max.6.in`, `max.6.out`

Le but de ce test est de vérifier que le programme fonctionne bien avec 20 000 000 d'événements générés par 1 500 capteurs. Les fichiers d'entrée et de sortie peuvent être générés par le script Python détaillé dans le préambule. Il ajoute 20 000 000 d'événements, puis vérifie que les commandes `STATS_C`, `JAM_DH` et `STATS_D7` renvoient les résultats attendus.

Avertissement : Le fichier d'entrée peut avoir une taille d'environ 500 Mo si 20 000 000 d'événements sont générés.