Compte rendu TP Trafic de Lyon - C++

Spécification

*I. Choix généraux*

Nous avons choisis une solution intermédiaire entre un stockage des données brutes uniquement et une méthode orientée Big Data où l’on ne stockerait que les résultats des requêtes. En effet ici nous ne nous intéressons qu’à l’intervalle de temps où un capteur est dans un état donné et non à ses états successifs. Ceci signifie que nous ne stockons pas les valeurs envoyées par le capteur mais nous en déduisons plutôt la durée durant laquelle le capteur est dans un état donné. Ce léger calcul est peu couteux en CPU et nous évitera ultérieurement de parcourir un trop grand nombre de données.

De plus, nous stockons et actualisons également le résultat de la requête ‘MAX\_TS’ (le moment avec le plus de bouchons simultanés). Nous avons fait ce choix car cette requête était particulièrement gourmande en temps de calcul par rapport aux autres requêtes. De cette manière nous pensons avoir amélioré les performances de notre application.

*II. Structures de données*

*III. Description technique*

*N.B. (1): les méthodes privées seront placées après les méthodes publiques et seront en italique.*

*N.B. (2): Etant donné le nombre important de méthodes et le nombre de classes de notre projet, les attributs et méthodes les triviaux ne seront pas expliqués. De même, les classes ‘Date’ et ‘ArbreIdentifiants’ sont spécifiées succinctement car plutôt classiques. Vous pourrez dans les deux cas vous référer au différents headers qui sont eux bien plus complets.*

***III.1. Classe ‘Main’***

*Rôle de la classe*

Gère le lancement de l’application ou bien des tests.

*Les méthodes*

. **static void LancementApplication**()

**. static** **void** **LancementTests**()

**. int** **main**()

* Appelle le Lancement de l’application ou bien des tests.

***III.2. Classe ‘GestionTrafic’***

*Rôle de la classe*

Interprète les commandes données par l'utilisateur en lignes de commandes et appelle les méthodes nécessaires dans les autres classes. Cette classe gère donc l'ensemble du trafic de Lyon.

*Les méthodes/constructeurs/destructeurs*

**. void** **Start** ()

* Identifie la commande donnée par l'utilisateur en ligne de commande grâce à la méthode 'determineTypeCommande' (qui renvoie une constante entière) et appelle la méthode correspondante à la constante.

**. void** **Start**(string commande)

Identifie la chaine de caractères donnée en paramètre grâce à la méthode 'determineTypeCommande' (qui renvoie une constante entière) et appelle la méthode correspondante à la constante.

**. GestionTrafic**()

* Initialise EvenementsCapteur (constructeur par défaut de Evenements pour initialiser 'toutLeTrafic')

**. virtual** **~GestionTrafic**()

* supprime et libere l'espace memoire correspondant a GestionTrafic.

**. *void******appelerAjouter****(string idCapteurReel,string annee, string mois, string jourDuMois, string heure, string minute, string seconde, string jourDeLaSemaine, string trafic)*

***. void******appelerStatistiquesCapteur****(string idCapteurReel)*

***. void******appelerStatistiquesJourSemaine****(string nJour)*

***. void******appelerStatistiquesJourHeureSemaine****(string nJour, string heure)*

***. void******appelerMaxBouchonsSimultanes****()*

***. int******determineTypeCommande****(****const*** *string &commande)*

*Ses attributs*

. Evenements \*toutLeTrafic;

***III.3. Classe ‘Evenements’***

*Rôle de la classe*

Gère l'ensemble des EvenementsCapteur en stockant les 1500 pointeurs vers les 1500 EvenementsCapteur potentiels dans un tableau. Si l'utilisateur ajoute une nouvelle donnée, alors cette classe la redirigera vers l'EvenementsCapteur associé.

*Les méthodes/constructeurs/destructeurs*

**. void** **AjouterEvenement**(**int** idCapteur, **int** heure, **int** jourSemaine, **int** trafic, **int** anneeEvent , **int** moisEvent, **int** nJourMoisEvent, **int** minutesEvent, **int** secondesEvent)

* Permet d'ajouter une donnée:

-met à jour 'capteurs' en ajoutant (méthode 'ajouter') la donnée à insérer ('idCapteur' et 'trafic' en paramètres)

-met à jour 'dateDernierEvenementTrafic' en remplacant cette date par la date de l'evenement à insérer ('anneeEvent', 'moisEvent', 'nJourMoisEvent', 'jourSemaine', 'heure', 'minutesEvent', 'secondesEvent' en parametres)

. **void AfficherTousLesEvenements**()

**. void** **StatistiquesCapteur**(**int** idCapteurReel)

* Appelle la méthode 'StatistiquesParCapteur' pour l'EvenementsCapteur se trouvant dans la case de 'capteurs' pour le capteur correspondant à ‘idCapteurReel’. On trouve la position de ce capteur dans le tableau grâce à sa position dans 'arbreId'.

**. void** **StatistiquesJourSemaine**(**int** nJour)

* Affiche les pourcentages des temps passés dans chaque état pour un jour de la semaine 'nJour' sur l'ensemble des capteurs.

**. void** **StatistiquesJourHeureSemaine**(**int** nJour, **int** heure)

* Affiche les pourcentages des temps passés dans chaque état pour un jour de la semaine 'nJour' et une heure donnée 'heure' sur l'ensemble des capteurs.

**. int** **AjouterIdAArbre**(**int** idCapeurReel)

**. Evenements**()

* Initialise EvenementsCapteur :

-'ArbreIdentifiants' initialisé à un arbre vierge (constructeur par défault)

-Toutes les cases de 'capteur' sont initialisées à 0

-'BouchonsMax' est initialisé à 0

**. virtual** **~Evenements**()

* Supprime et libère l'espace mémoire correspondant à l'Evenements

*Ses attributs*

. ArbreIdentifiants \*arbreId

. EvenementsCapteur \*\*capteurs

* Tableau de 1500 case qui contient des ‘EvenementsCapteurs’

. Date dateDernierEvenementTrafic

. Date dateBouchonMax

**. float** bouchonMax

***III.4. Classe ‘EvenementsCapteurs’***

*Rôle de la classe*

Gère les données relatives à un capteur en stockant le nombre de secondes passées pour un jour, une heure et un état particulier. Stocke également la date du dernier évenement pour ce capteur et le dernier état du trafic pour ce capteur.

*Les méthodes/constructeurs/destructeurs*

**. void** **Ajouter**(**int** trafic, Date date)

* Incrémente la durée d'un état de trafic (en paramètre 'trafic') dans 'secondesPassees' à la durée qui était déjà existante. Cette incrémentation se fait à la Date correspondante au paramètre 'date' (case heure, jour de la semaine, état du trafic).

Si la durée d'incrémentation est supérieure à 5 min, alors 5 min sont ajoutées.

Si la durée ajoutée est sur 2 jours ou sur 2 heures, alors l'incrémentation se fait sur 2 cases différentes de 'secondesPassees'.

. void **Afficher**();

**. void** **StatistiquesParCapteur**()

* Affiche le pourcentage de temps passé dans chaque état.

**. double**\* **SecondesPasseesDansChaqueEtat**(**int** jour, Date dateDernierEvenementTrafic)

* Permet d'obtenir le temps passé dans chaque état pour un jour de la semaine 'jour', et la date du derner evenement ajouté 'dateDernierEvenementTrafic' en paramètres.

Retourne un tableau de pointeurs vers des double. Chaque case du tableau pointe vers des valeurs correspondantes au temps passé dans un état donné.

**. double**\* **SecondesPasseesDansChaqueEtat**(**int** jour, **int** heure, Date dateDernierEvenementTrafic)

* Permet d'obtenir le temps passé dans chaque état pour un jour de la semaine 'jour', pour une heure de la journée 'heure', et la date du derner evenement ajouté 'dateDernierEvenementTrafic' en paramètres.

Retourne un tableau de pointeurs vers des double. Chaque case du tableau pointe vers des valeurs correspondantes au temps passé dans un état donné.

**. EvenementsCapteur**(**int** trafic, Date date)

* Initialise EvenementsCapteur :

-'isEmpty' initialisé à false

-'traficDernierEvenement' est initialisé à la valeur de 'trafic'

-'secondesPassees' a tous ses pointeurs initialisés à 0

-'dateDernierEvenement' est initialisé à 'date'

**. EvenementsCapteur**()

* Initialise EvenementsCapteur :

-'isEmpty' initialisé à true

-'traficDernierEvenement' est initialisé à 0

-'secondesPassees' a tous ses pointeurs initialisés à 0

**. virtual** **~EvenementsCapteur**()

* supprime et libère l'espace mémoire correspondant a l'EvenementsCapteur.

***. int******max5minutes****(****int*** *nombreSecondes)*

* *Permet de savoir combien de temps doit être ajouté à la durée d'un evenement.*

*Retourne nombreSecondes si nombreSecondes est inférieur à 5 min.*

*Retourne nombreSecondes si nombreSecondes est supérieur à 5 min.*

*Ses attributs*

*.*Date dateDernierEvenement

**.int** traficDernierEvenement

**.bool** isEmpty

**.int** \*\*\*secondesPassees

* Tableau d’entier à 3 dimensions. Chaque case rèfere à une heure de la journée, un jour de la semaine et un état du trafic.

***III.5. Classe***

*Rôle de la classe*

*Les méthodes/constructeurs/destructeurs*

*Ses attributs*

***III.6. Classe ‘Date’***

*Rôle de la classe*

La classe ‘Date’ gère une date sous forme d'une année, un mois, un jour du mois, un jour de la semaine, une heure, une minute, une seconde et un nombre de secondes depuis le début de l'année. On peut notamment comparer, additionner ou encore soustraire des dates grâce à des surcharges d’opérateurs.

***III.7. Classe ‘ArbreIdentifiants’***

*Rôle de la classe*

Cette classe ordonne les identifiants réels de capteurs sous la forme d'un arbre. La position dans l'arbre de l'identifiant va devenir un nouvel identifiant pour chaque capteur. Ces nouveau identifiants seront tous les entiers compris entre 0 et 1499. Cela permettra de déterminer et pouvoir récupérer la position d'un capteur dans un éventuel tableau.

***III.8. Classe TestsGestionTrafic***

*Voir document de description des tests fonctionnels.*

***III.9. Le header ‘Constantes’***

*Rôle du header*

Stocke les constantes utilisées fréquemment dans tout le code de l’application.