TP C++ 2

Spécification et choix généraux

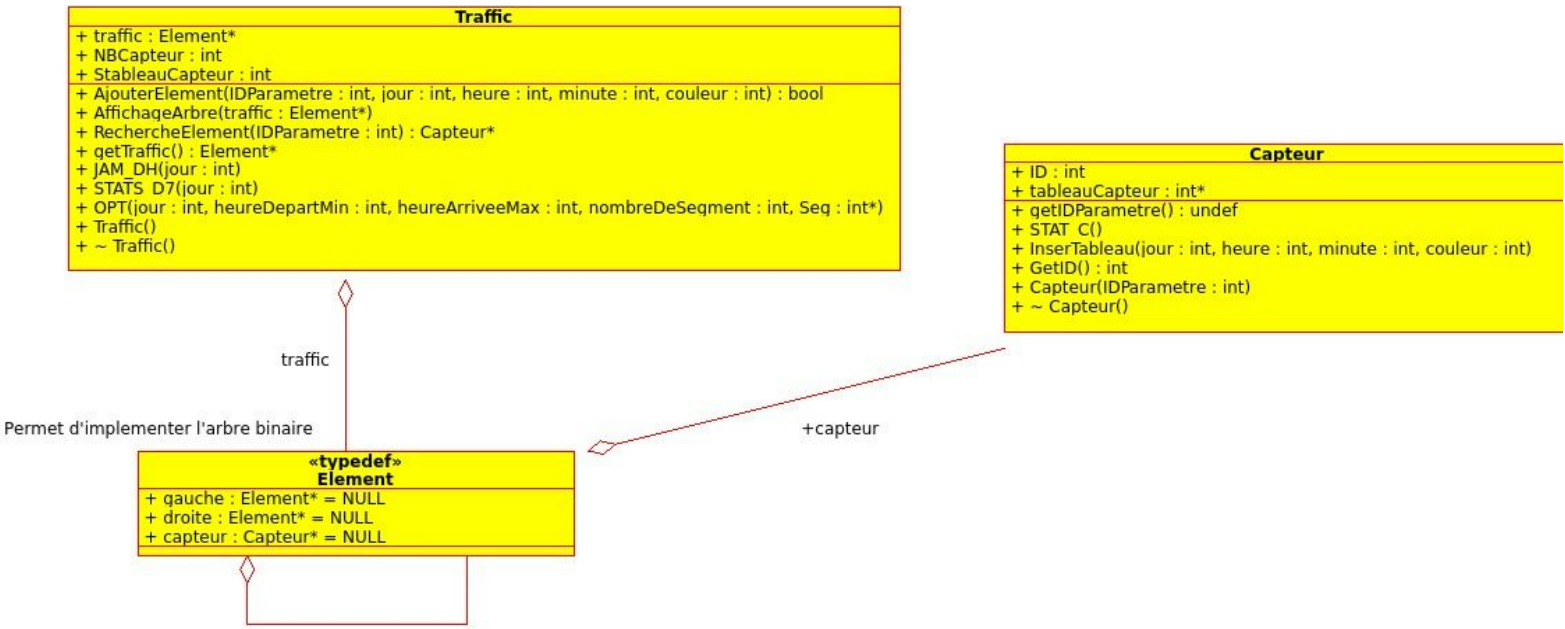
Jacques FOLLEAS Arnaud DUPEYRAT

Sommaire:

1. INTRODUCTION

Le role de ce programme est de restituer des statistiques du trafic routier de la ville de Lyon.
Pour cela, dans un premier temps, le programme doit acquerir des mesures de capteurs qui fournissent les informations du trafic routier.
Dans un second temps, grace a une serie de commandes, le programme renvoie des statistiques correspondants aux mesures réalisées.
Ce programme doit répondre a des contraintes de performance importante pour etre le plus efficace possible.

2. Description générale de notre programme et principales définitions.
    a. Diagramme des classes



b. Choix généraux

- Nous avons fait le choix de créer deux modules pour répondre au probléme:
    - La classe **Capteur** regroupe les données d'un capteur.
    - La classe **Traffic** contient l'ensemble des données à étudier.

  Ces deux classes sont reliées à l'aide d'un arbre binaire dans lequel chaque noeud contient les informations collectées par un capteur.
  (cf diagramme).

- La gestion du flux de données se fait par l'intermediaire des objets cin/cout, permettant de gérer différents types de données facilement. Nous avons valide que ce choix nous permettait d'atteindre pas les performances ciblées par le cahier des charge.

- Le choix de l'arbre binaire a été motivé par le nombres d'évenements important à traiter car il permet d'acélerer la recherche d'élement et est adapté à un traitement d'IDs aléatoire. Cela permet une recherche en temps inferieur à 0(n).

- Pour alléger le nombre de données à conserver, nous ne conservons ni le mois, ni l'année, ni le jour des différentes mesures enregistrées.

- Le cahier des charges n'était pas précis sur la description des valeurs de retour dans le cas ou aucune information concernant le capteur pour un jour donnée n'avait été reçu.
  Dans cette situation nous retournons alors pour toutes les méthodes des valeurs nulles.

- Dans la méthode OPT nous considérons par défaut le trafic comme fluide (V) si aucune information concernant le capteur n'est recensé dans le données reçues.

3. Description des différentes Modules composant le programmes.
    a. Classe Traffic
        ii. Preambule

Le **rôle** de la classe Traffic est double:

- Premièrement elle gére la structure de données et permet l'insertion des données.
- Deuxièmement permettre le fonctionnement des differentes **méthodes** concernant l'ensemble des capteurs.
    - JAM_DH()
    - STATS_D7()
    - OPT()

Les **attributs** de la classe sont les suivants:

- **traffic** permettant d'accéder à la racine de l'arbre binaire.
- **NBCapteur** donnant le nombre de capteur présent dans la structure de données.

- **STableauCapteur** tableau permettant d'insérer des données concernant l'ensemble des capteurs.

iii. Implementation Arbre binaire, méthode AjouterElement().

Cet arbre binaire permet de répondre de maniére éfficace aux demandes de l'utilisateur. Pour cela plusieurs méthodes ou struct ont été utilisées :

- **une struct (typedef Element)** est composé d'un Capteur * capteur, d'un Element * droite, et d'un Element * gauche. struct nous permettant d'implémenter notre structure de données.

- **Une méthode CreationElement(int ID)**
Cette méthode initialise un Element de la structure de données en fonction des valeurs entrées par l'utilisateur. Elle crée une instance de Capteur avec le nouveau ID et initialise les pointeurs droite et gauche à des valeurs NULL.

- **Une méthode AjouterElement (int ID, int jour, int heure, int minute, int couleur)**
La méthode AjouterElement() permet d'ajouter une **mesure** correspondant a un capteur. Son implementation gère trois cas différents en fonction de l'état du trafic:
    1) Si c'est la toute première mesure, la méthode relie alors la racine (traffic) au premier Element crée.
    2) Si L'ID du capteur reçu existe deja, alors nous cherchons ce capteur afin de modifier les données concernant le capteur.
    3) Si le capteur n'est pas présent au sein de la structure, il est necessaire de l'insérer à la bonne place pour une recherche optimisé par la suite. Dans cette optique nous comparons l'ID des différents capteurs,
        - si L'iD du capteur est superieur au capteur de l'element courant alors nous nous "déplaçons" sur la droite
        - si ce n'est pas le cas nous nous "deplaçons" sur la gauche.

    De plus dans tous les cas, la méthode est en charge d'insérer les valeurs utiles pour la suite du programme : Elle ajoute les nouvelles valeurs à TableauCapteur et elle modifie le StableauCapteur regroupant la somme des valeurs permettant de définir l'activité du traffic.

- **Une méthode RechercheElement (int ID)**
Cette méthode renvoie un Capteur * associé à l'ID passé en paramétre.

- **Une méthode AfficherArbre (Element* traffic)**
Elle permet l'affichage de notre structure de donnés par ID croissant. AfficherArbre fonctionne de maniére recursive. (Méthode non utilisé dans la suite du programme mais utile pour vérifier l'état de notre structure de données.)

iv. Méthode JAM_DH

Cette méthode retourne pour un jour de la semaine le pourcentage d'embouteillage par heure. Elle calcule le pourcentage de couleur rouge et noire mesuré pour l'ensemble des capteurs présent sur le trafic pour chaque heure du jour de la semaine passé en paramêtre. Elle permet de visualiser la distribution par heure du trafic sur une journee particulière.

v. Méthode STATS_D7

Cette méthode a pour but d'afficher un compte rendu de l'état d'un trafic pour un jour de la semaine. Elle permet de de connaitre la distribution de l'état du trafic pour un jour donné en calculant la moyenne par couleur de trafic (Vert, Jaune, Rouge, Noir).

vi. Méthode OPT

Cette méthode permet d'afficher l'heure du dernier départ permettant de parcourir un ensemble de capteur avant une heure d'arrivé donnée: heureArriveeMax.
On considère qu'il faut 1 minute pour traverser un segment associé à une couleur V, 2 pour J, 4 pour R, et 10 minutes pour N. Pour calculer le temps nécessaire pour parcourir un segment nous avons éffectué la moyenne pondéré des temps par couleur variant chaque minute. Ainsi il nous a été possible de connaitre le temps nécessaire pour parcourir chaque segment.
Si selon les statistiques du trafic, il n'est pas possible d'arriver à l'heure, le programme renvoie alors un message d'erreur à l'utilisateur.

f. classe Capteur

i. Spécification générale

Le rôle de la classe est de modéliser le fonctionnement d'un capteur appartenant au traffic.

La classe posséde plusieurs **attributs** :

- **ID** attribut unique,identifiant le capteur.
- **TableauCapteur** Un tableau contenant les mesures collectes par le capteur.

Cette classe contient la methodes suivante:

ii. Méthode STATS_C()

Cette methode retourne les informations correspondant au trafic d'une instance de capteur.
STAT_C renvoie le pourcentage selon chaque couleur, en se basant sur l'ensemble des données.

g. main.cpp

i. Spécification gérérale

Le programme principale permet de faire le lien entre les commandes entrées par l'utilisateur et la structure de données representant le trafic de la ville lyonnaise.
Le programme entre alors dans une boucle de reconnaissance de textes d'entrée traduits en commande au programme. Celui-ci se termine seulement si elle reconnait la commande « EXIT ».
Les mots clés reconnus sont:

- La commande **« ADD »** qui fait appel à la fonction AjouterElement (cf spécification Traffic).
- La commande **« STATS_C »** retourne les statistiques concernant le capteur recherché.
- La commande **« JAM_DH »** retourne les statistiques sur les embouteillages par jour de la semaine et par heure.
- La commande **« STATS_D7 »** retourne les statistiques decrivant l'etat du trafic correspondant au jour demandé.
- La commande **« OPT »** permet d'afficher le moment de départ optimale qui minimise le temps passé sur le parcours.

D'autre part nous avons testé la performance de notre programme à l'aide de la librairie time.h, permis par le cahier des charges.
Il nous a alors été possible d'obtenir une estimation du temps nécessaire au programme pour répondre à des demandes importantes de l'utilisateur (cf specification Test).
La partie concernant les performances est en commentaire dans le programme afin d'obtenir des reponses .out conforme à la réponse attendue par le cahier des charges.

h. Tests.cpp

i. Spécification générale

Pour valider notre programme nous avons du implementer des tests contenant un nombre de données important, c'est à dire 1500 capteurs, 20 000 000 d'évenements.
Pour répondre à ce problème, il nous a fallu construire un module tests.cpp permettant à l'aide des redirection de flux de creer un fichier.in contenant un nombre de données assez important pour finalement mettre à l'épreuve notre programme.
Nous sommes conscients que le cahier des charges ne permettait pas d'utiliser des librairies tel que fstream mais l'utilisation de cette derniere nous semblait indispensable pour gérérer des données de l'ordre 20 000 000 d'événements ou de l'ordre de 1500 capteurs.