IF-3-C++-POO2-TP N°4

Analyse de logs apache

Document de conception

Table des matières

Table des illustrations

IF-3-C++-P002-TP N°4
Analyse de logs apache

Document de conception

I. Introduction

Pour ce quatrième et dernier TP de « Programmation Orientée Objet » dans le langage C++, l'objectif est de se concentrer sur la **STL**, et plus particulièrement en relation avec **un fichier de logs apache** à analyser via une application utilisable en ligne de commande. Plus concrètement, ce TP nous permet d'approfondir différents **conteneurs** et **algorithmes** de la STL. Pour ce faire, nous allons étudier un fichier log provenant de **l'intranet IF de l'INSA de Lyon**, et en dégager des **statistiques** sur les **10 pages les plus consultées** ainsi qu'un graphique sous le format ***GraphViz***. Ce document de conception va reprendre de manière concise **les spécifications générales** de notre application, ainsi que des **spécifications plus détaillées**. Nous présenterons aussi l'architecture globale de notre application via un diagramme **UML**. Pour terminer, nous évoquerons la **structure de données** utilisée.

II. Spécifications générales

II.1. Utilisation générale

L'outil d'analyse ***analog*** est un outil en ligne de commande permettant d'analyser un journal de logs apache. Cet outil permet de lire au travers d'un grand fichier de log, et d'en extraire des objets de type *LogElement*. Ces éléments contiennent la **totalité des informations lisibles** dans un fichier de **logs apache**, de manière organisée dans différents attributs, pour les utiliser plus aisément. Cet outil permet, via différentes options, d'en ressortir les **10 cibles les plus touchées par les requêtes** ainsi qu'un fichier **.dot** sous le format *GraphViz*, **transformable par la suite en graphique**, représentant les **interactions** entre **cible** et **référant**. Dans ce dernier, chaque URL (cible ou source) connu est représenté par un **sommet**, et des **arcs** relient ces derniers entre eux, représentant le nombre de **hits**. Différentes options permettent de **filtrer les logs** pour ne pas prendre en compte la totalité des données. Plus simplement, les options permettent d'affiner l'utilisation de l'outil *analog*.

Ces options sont présentées et décrites dans le manuel d'utilisation fourni.

On rappelle toutefois la syntaxe générale de l'outil : **./analog [option] logFile.log**

II.2. Choix d'implémentation réalisés

- ***Implémentation des options :***

L'implémentation a été faite telle que **l'ordre des options n'importe pas**. Il en est de même pour le nom du fichier log, pouvant se trouver entre 2 options, même au tout début après l'appel de l'outil *analog*. Toutefois, **les options doivent respecter une certaine syntaxe** pour être autorisée par l'outil, car ce dernier vérifie **scrupuleusement** leur **composition** avant de le lancer réellement.

> ***[-g graphName.dot]*** – graphName.dot ne doit pas exister et contenir l'extension .dot

> ***[-e]***

> ***[-t heure]*** – le paramètre heure doit être un entier compris entre 0 et 23.

De plus, l'outil **ne pourra pas être lancé** via l'option -g **si un fichier du même nom avec l'extension .dot existe déjà**, de manière à éviter la **suppression malencontreuse** d'une analyse précédente.

- ***Nettoyage du fichier de log :***

Le fichier de log étant assez complexe, il était important d'effectuer quelques **nettoyages** pour le **traiter efficacement**. Nous avons donc faire certains choix pour implémenter notre outil :

Tout d'abord, nous **supprimons l'URL de base du serveur de logs apache** s'il existe dans le référant de la requête. Dans notre cas, il s'agit de http://intranet.if.insa-lyon.fr. Il n'est pas codé en « dur » dans le programme, et est **modifiable** durant la **construction** d'une **instance** de *InputLogStream.*

Ensuite, certains **référant** sont **inconnus**, et sont donc égaux au caractère « **-** ». Nous avons donc fait le choix de **retirer ces lignes de logs durant la création du graphe** pour qu'elles ne viennent pas **perturber** le **graphique**. Toutefois, on les **conserve** pour les **stats**.

De la même façon, nous **pourrions** retirer les requêtes dont le code de retour est **404** (page inexistante). Toutefois, nous ne l'avons pas implémenté ici, car cela reste facultatif et peu utile pour notre cas.

- ***Implémentation du résultat :***

Nous avons fait le choix, **peu importe les options choisies**, d'afficher le **top 10** des cibles en premier. **Si moins de cibles sont disponibles, on en affiche simplement moins de 10**.

II.3. Spécifications détaillées et jeux de tests

- ***Cas d'erreurs :***

Cas d'utilisation - Erreurs	*Que doit-il se passer ?*	*Test(s) n°*
Pas de fichier log passé en argument	***[ERROR] Syntax : Use ./analog [-g graphName.dot] [-e] [-t time] fileName.log*** *Arrêt de l'application avec en retour un code d'erreur 1*	***1***
Le fichier de log passé en argument est non existant	***[ERROR] Log-file : Invalid or inexistant log-file - Can't open it.*** *Arrêt de l'application avec en retour un code d'erreur 1*	***2***
Le fichier de log est sans droit en lecture	***[ERROR] Log-file : Invalid or inexistant log-file - Can't open it.*** *Arrêt de l'application avec en retour un code d'erreur 1*	***3***
Option -g : fichier .dot déjà existant dans le dossier	***[ERROR] Dot-file : Dot-file already exist.*** *Arrêt de l'application avec en retour un code d'erreur 1*	***4***
Mauvaise syntaxe de manière générale, option inexistante ou non reconnu.	***[ERROR] Syntax : Use ./analog [-g graphName.dot] [-e] [-t time] fileName.log*** *Arrêt de l'application avec en retour un code d'erreur 1*	***5***
Mauvaise syntaxe : option -g sans préciser de fichier .dot à la suite	***[ERROR] Syntax : Option -g not followed by a dot file. Use ./analog [-g graphName.dot] [-e] [-t time] fileName.log*** *Arrêt de l'application avec en retour un code d'erreur 1*	***6***
Mauvaise syntaxe : option -t sans préciser l'heure à la suite	***[ERROR] Syntax : Option -t not followed by an int. Use ./analog [-g graphName.dot] [-e] [-t time] fileName.log*** *Arrêt de l'application avec en retour un code d'erreur 1*	***7***
Mauvaise syntaxe : répétition d'une des options -t, -e ou -g dans la même commande	***[ERROR] Syntax : You can't repeat option {-t,-e,-g} in analog command. Use ./analog [-g graphName.dot] [-e] [-t time] fileName.og*** *Arrêt de l'application avec en retour un code d'erreur 1*	***8, 9, 10***
Option -t : heure non cohérente, c'est-à-dire pas dans l'intervalle [0,23].	***[ERROR] Syntax : Invalid time - Parameter must be an integer in [0,23]*** *Arrêt de l'application avec en retour un code d'erreur 1*	***11***

- ***Cas d'exécution sans erreurs :***

A présent, chacun des cas ci-dessous est garantie sans erreurs provenant d'un des cas ci-dessus.

Cas d'utilisation – Sans erreurs	*Que doit-il se passer ?*	*Test(s) n°*
Sans aucune option : **./analog file.log**	***Affirmation qu'aucun problème ne s'est produit : Analog launched without errors.*** Aucun problème. Exécution de l'outil avec ses sorties standards de résultats contenant le top 10.	***12***
Utilisation de l'option -g	***Affirmation qu'aucun problème ne s'est produit : Analog launched without errors.*** Aucun problème. Exécution de l'outil avec ses sorties standards de résultats contenant le top 10 et production du fichier .dot avec avertissement : Dot-file dotName.dot generated	***13***
Utilisation des options -e et -g	***Affirmation qu'aucun problème ne s'est produit : Analog launched without errors.*** Aucun problème. Exécution de l'outil avec ses sorties standards de résultats contenant le top 10 et production du fichier .dot en ignorant les cibles provenant d'images, css et javascript avec un avertissement en plus du cas précédent : Warning : hits and refs on image, css or javascript extensions have been not taken into account	***14***
Utilisation des options -e, -g et -t	***Affirmation qu'aucun problème ne s'est produit : Analog launched without errors.*** Aucun problème. Exécution de l'outil avec ses sorties standards de résultats contenant le top 10 et production du fichier .dot en ignorant les cibles provenant d'images, css et javascript avec avertissement et en ne gardant que les logs du créneau [T, T+1[avec un avertissement en plus du cas précédent : Warning : only hits between Th and (T+1)h have been taken into account	***15***
Utilisation de l'option -e	***Affirmation qu'aucun problème ne s'est produit : Analog launched without errors.*** Aucun problème. Exécution de l'outil avec ses sorties standards de résultats contenant le top 10 en ignorant les cibles provenant d'images, css et javascript avec avertissement comme précédemment.	***16***
Utilisation de l'option -t	***Affirmation qu'aucun problème ne s'est produit : Analog launched without errors.*** Aucun problème. Exécution de l'outil avec ses sorties standards de résultats contenant le top 10 en ne gardant que les logs du créneau [T, T+1[avec avertissement comme précédemment.	***17***
Utilisation des options -g et -t	***Affirmation qu'aucun problème ne s'est produit : Analog launched without errors.*** Aucun problème. Exécution de l'outil avec ses sorties standards de résultats contenant le top 10 et production du fichier .dot en ne gardant que les logs du créneau [T, T+1[avec avertissement comme précédemment.	***18***
Utilisation des options -e et -t	***Affirmation qu'aucun problème ne s'est produit : Analog launched without errors.*** Aucun problème. Exécution de l'outil avec ses sorties standards de résultats contenant le top 10 en ignorant les cibles provenant d'images, css et javascript et en gardant que les logs du créneau [T, T+1[avec avertissement comme précédemment.	***19***
Sans aucune option avec un fichier log vide	***Affirmation qu'aucun problème ne s'est produit : Analog launched without errors.*** Aucun problème. Exécution de l'outil avec ses sorties standards de résultats contenant un message d'avertissement et un top 10 précisant qu'une requête du log n'a été récupéré sur le fichier de log vide : Warning : Log-file generate 0 log-line with the actual analog configuration.	***20***

III. Architecture globale de l'application

Voici le diagramme de classe UML reprenant clairement l'architecture globale de l'application.

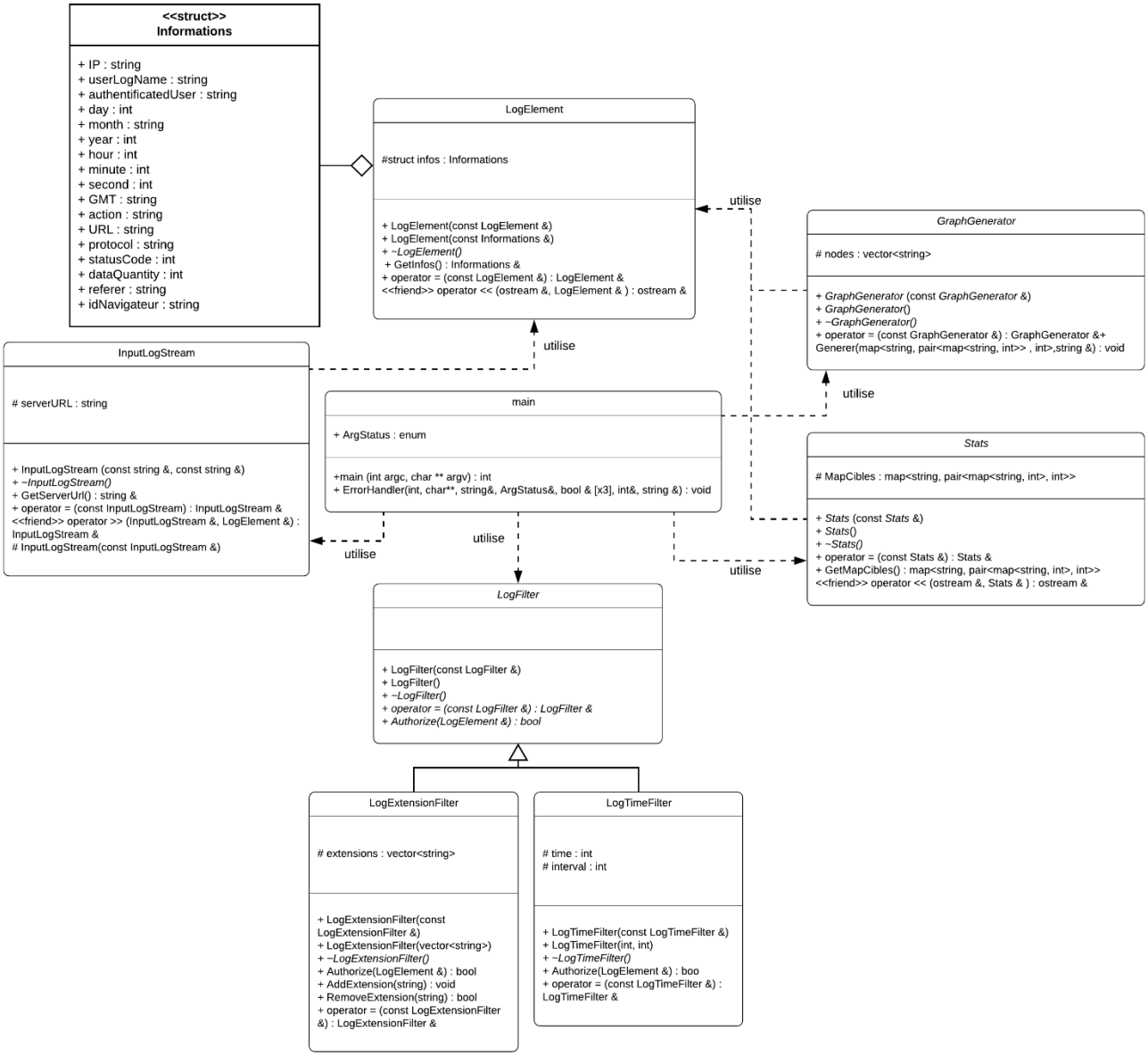

Figure 1 : Diagramme UML de notre outil analog

Nous avons fait le choix d'encapsuler notre application dans **7** classes différentes en plus du *main* (point d'entrée de l'application). Ce dernier gère notamment les **différentes erreurs** dû à l'appel de l'outil *analog* via sa méthode ***ErrorHandler***. Ensuite, une première classe ***LogElement*** contient les différentes informations que l'on retrouve dans une ligne de log apache. Puis, une classe ***InputLogStream*** hérite de ***ifstream*** de façon à récupérer le fichier .log en entrée, et de le lire ligne par ligne

via la **surcharge de l'opérateur >>.** D'ici, on passe les logs obtenus dans **aucun, un ou deux filtres** en fonction des options choisies. Nous avons la classe abstraite ***LogFilter*** qui permet de créer un **socle de base** pour nos deux filtres (de façon à les gérer dans une collection hétérogène), de là hérite ***LogExtensionFilter*** et ***LogTimeFilter***, respectivement le filtre d'extensions pour l'option ***-e*** et le filtre de temps pour l'option ***-t.*** Les logs ainsi filtrés se retrouvent dans la classe ***Stats*** où ils sont stockés dans une structure de données, qui sera détaillé plus tard, d'où l'on en tire un **classement**, ainsi qu'un graphique sous le format *GraphViz* grâce à la classe ***GraphGenerator.***

IV. Structure de données employée

Comme présenté dans le diagramme UML précédent, nous avons fait le choix d'utiliser, pour notre structure de données principale, une **MAP** de la STL construite d'une façon plutôt complexe.

En effet, notre choix final s'est tourné sur une : ***map<string, pair<map<string, int>, int>>.***

Tout d'abord, intéressons-nous aux différentes parties présentes dans cette structure de données : la première ***map***, que l'on appellera ***map principale*** contient comme **clé** un **string**. C'est le **nom des différentes cibles** que l'on trouve dans un fichier .log. Ainsi, chacune de ces cibles est alors liée avec, comme **valeur**, une ***paire***. Nous voulions au départ utiliser une classe bien distincte, mais finalement, nous n'avions que deux attributs présents dans cette classe, et rien d'autre. Nous avons donc fait le choix d'implémenter cette « classe » sous la forme d'une **paire**. Cette dernière contient comme ***first*** attribut une nouvelle ***map***, que l'on nommera ***map secondaire***, qui contient quant-à-elle **chaque référent** de la cible associée, ainsi que le **nombre de hits** associé entre les deux. La clé de cette *map* est donc le **string** nommant les référents, et **l'entier** correspond au **nombre de hits**. Enfin, le *second* attribut de la paire est un **entier**, qui correspond au **nombre de hits total de la cible associée.**

Pourquoi avoir fait ce choix ? Dans un premier temps, le cahier des charges nous demandait la récupération d'un classement des 10 cibles les plus touchées. Pour récupérer cette information d'une façon rapide, nous avons choisi d'utiliser la **map principal**. Effectivement, chaque cible est clé de la map, elle est **unique** grâce à l'implémentation de cette dernière. Il est ainsi très simple de venir chercher la clé puis d'incrémenter son nombre de hits à chaque ajout d'un élément de Log. **L'insertion se fait en o(log(n)) et la recherche de même**. Au vu du grand nombre de données que l'on peut

récupérer (par exemple un fichier .log de 100 000 lignes), le **log** est **extrêmement efficace** ici. De ce fait, nous pouvons récupérer en une dizaine de parcours les 10 cibles les plus touchées. Dans un second temps, nous devions être capable de **créer**, via l'option -g, un **graphique** sous le format ***GraphViz***, celui-ci nécessite de stocker aussi les référents de chaque cible, et leur nombre de hits associés. Là est l'intérêt de la **map secondaire**, qui est associé à chaque cible et contient ainsi chaque référent associé. Pour la création du graphique, il suffit de parcourir chaque cible de la map principale, et d'y récupérer chaque référent avec son nombre de hits. **Encore une fois, le parcours en o(log(n)) nous est extrêmement utile.**

Voici une représentation de la structure de données utilisée, en sachant qu'une map n'est rien d'autre qu'un arbre rouge et noir dans la STL.

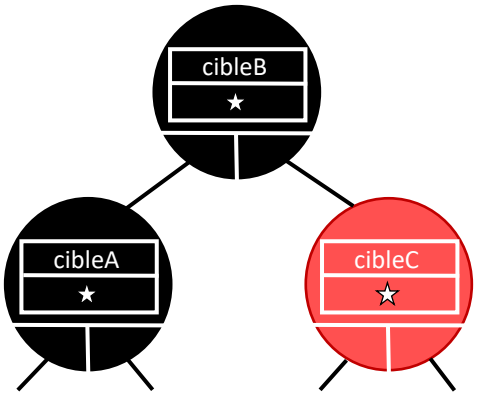

Figure 2 : Schéma de l'implémentation de notre arbre principal

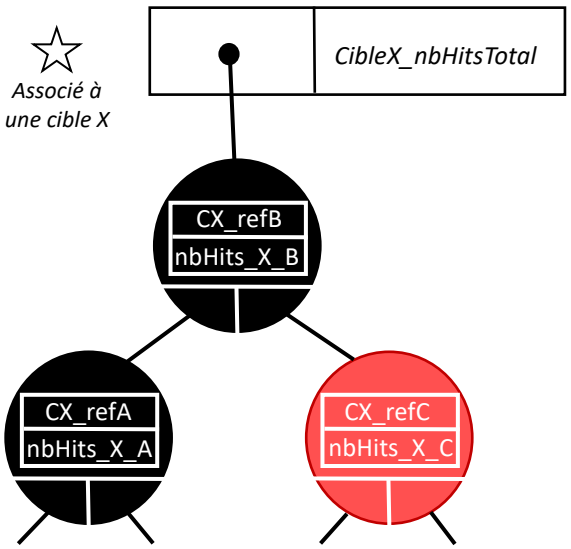

Figure 3 : Schéma de l'implémentation de la valeur de l'arbre principal contenant l'arbre secondaire et un entier.

V. Conclusion

Le plus gros objectif de ce TP était de compléter nos connaissances sur quelques **briques de base** de la STL. Nous pensons qu'il a clairement été complété, et nous a permis de nous **sensibiliser à l'importance de la complexité** dans le choix de nos structures de données. Encore plus pour des quantités importantes de données à gérer. De plus, implémenter un cahier des charges en gardant en tête la **réutilisabilité des classes** était aussi très intéressant pour nos futurs projets.