TP4 - Analyse de logs Apache

Document de Conception

I. Spécifications

I.A. Spécifications générales de l'application

Le programme réalisé (*analog*) est un outil en ligne de commande permettant d'analyser un fichier de log de serveur web Apache. L'outil *analog* voit un fichier de log Apache comme un ensemble de requêtes HTTP adressant une URL cible à partir d'une URL source (le *referer*). Il a deux fonctionnalités majeures : trier et afficher les URL cibles par popularité (c'est à dire par nombre décroissant de requêtes les adressant), et générer un fichier au format GraphViz pour l'exploitation du fichier de log sous forme de graphe, où chaque URL connue différente (cible ou source) est considérée comme un sommet différent, et chaque requête dont on connaît à la fois la source et la cible est un arc qui lie cette source à sa cible. Les arcs ne sont représentés qu'une seule fois, avec une étiquette qui indique le nombre de fois qu'ils apparaissent dans le fichier de log.

L'outil *analog* traite tous les types de méthodes HTTP de la même manière : il comptabilise toutes les tentatives d'accès à des ressources, qu'elles existent ou pas, peu importe le code de retour du serveur.

Les arguments passés en ligne de commande à *analog* permettent d'affiner son utilisation et décider quel traitement effectuer. Les arguments contiennent obligatoirement le chemin relatif ou absolu vers le fichier de log Apache à traiter, et facultativement des options. La syntaxe d'appel est la suivante :
$ ./analog [-e] [-t time] [-g gvfilename.dot] logfilename.log

Spécifications des options de sélection des requêtes du fichier de log :

- Option -t time : time est un entier compris entre 0 et 23. Sélectionne les requêtes dont l'heure de traitement est dans l'intervalle [time ; time+1]
- Option -e : Sélectionne les requêtes dont l'URL cible ne référence pas un fichier CSS (.css), Javascript (.js) ou une image (.png, .jpg, .jpeg, .bmp, .gif, .ico)

Les traitements effectués se déclinent en deux cas :

- *analog* n'est utilisé avec aucune option, dans ce cas la liste des 10 URL cibles les plus populaires dans le log sera affichée sur la sortie standard par ordre décroissant de popularité, avec le nombre de requêtes les adressant.
- *analog* est utilisé avec au moins une option, dans ce cas toutes les requêtes du fichier de log respectant tous les critères de sélection en option sont traitées. Si aucun critère de sélection n'est donné en option, toutes les requêtes sont traitées. La liste de toutes les URL cibles traitées sera affichée sur la sortie standard par ordre décroissant de popularité, avec le nombre de requêtes traitées les adressant. Si les options contiennent -g gvfilename.dot, un fichier au format GraphViz représentant le graphe que nous avons défini plus haut sera généré, et il aura pour nom gvfilename.dot. Il ne prend en compte que les requêtes traitées.

I.B. Spécifications détaillées et tests fonctionnels

I.B.1 Spécifications pour les arguments en ligne de commande

Les arguments optionnels peuvent être passés dans n'importe quel ordre au programme *analog*, mais le nom du fichier de log à analyser doit toujours être le dernier argument. La validité des valeurs passées par argument est vérifiée durant le programme.

Les caractères acceptés dans les noms de fichiers sont les caractères alphanumériques ainsi que les 4 caractères suivants : **/ . _ -** Les espaces sont interdits.

Cas d'utilisation	Action à effectuer	N° de Test
Arguments passés dans n'importe quel ordre	Rien à signaler, l'exécution se poursuit	**1, 2, 3, 4, 5, 6**
Arguments invalides. Syntaxe de passage des arguments non respectée. *Analog* n'a pas pu interpréter correctement l'ensemble des arguments passés, et n'a pas réussi à déterminer la cause de l'échec. Exemple : argument dupliqué, argument non reconnu	Ecriture dans la sortie d'erreurs : "Error : Invalid arguments passed to analog" "Usage : analog [-g gvfilename.dot] [-e] [-t time] logfilename.log" Terminer l'exécution avec le code 1	**7, 8, 9, 10**
Créneau horaire spécifié incorrect. Le paramètre time doit être un entier compris entre 0 et 23 inclus.	Ecriture dans la sortie d'erreurs : "Error : Invalid time value specified for parameter -t" Terminer l'exécution avec le code 1	**11, 12**
Nom de fichier GraphViz invalide. Le nom de fichier ne se termine pas par l'extension .dot ou comporte des caractères interdits.	Ecriture dans la sortie d'erreurs : "Error : Invalid filename specified for parameter -g" Terminer l'exécution avec le code 1	**13, 14**
Nom de fichier GraphViz déjà existant.	Ecriture dans la sortie d'erreurs : "Error : Specified output GraphViz filename already exists" Si non terminer l'exécution avec le code 1	**15**
Nom de fichier de log Apache invalide. Le nom de fichier log ne se termine pas par l'extension .log ou comporte des caractères non autorisés.	Ecriture dans la sortie d'erreurs : "Error : Invalid Apache Log filename" Terminer l'exécution avec le code 1	**16, 17**
Fichier de log Apache absent. Le fichier n'a pas pu être ouvert en lecture.	Ecriture dans la sortie d'erreurs : "Error : Apache Log file not found" Terminer l'exécution avec le code 1	**18**
Le fichier de log Apache spécifié est vide.	Ecriture dans la sortie standard : "Warning : Empty log file" "Nothing to be done" Aucun fichier n'est généré. Aucune autre sortie sur la sortie standard. Terminer l'exécution avec le code 0	**19**

I.B.2 Spécifications pour le traitement des requêtes et URL

L'outil *analog* fait subir à toutes les URL des traitements particuliers que nous spécifions ici :

- Si l'URL de base du serveur web Apache à l'origine du fichier de log (qui est http://intranet-if.insa-lyon.fr) apparait dans une URL source ou cible, cette URL de base sera supprimée, de même que le numéro de port correspondant à la connexion, s'il est spécifié.
- Toutes les URL sont passées en caractères minuscules.
- Les arguments des requêtes passés dans les URL sont également supprimés.

Nous définissons deux URL qui pointent vers la même ressource (source ou cible) comme deux URL qui sont identiques après ces traitements : elles représentent le même sommet.

Certaines URL ont des paramètres	Supprimer les paramètres des URL	**20**

Certaines URL ont des caractères majuscules	Passer ces URL en caractères minuscules	**21**
Certaines URL contiennent l'URL de base du serveur Apache	Supprimer l'URL de base du serveur Apache	**22**
URL présentées sous différentes formes Le fichier de log contient des URL qui pointent vers la même ressource mais qui se présentent différemment (casse différente, URL de base présente ou non, paramètres présents ou non)	Les URL sont nettoyées comme décrit ci-dessus, et doivent compter comme un unique sommet, auquel il est fait référence plusieurs fois.	**20, 21, 22**

I.B.3 Spécifications pour la production des sorties

Si la source d'une requête n'est pas connue, l'arc n'est pas généré pour l'exportation dans le fichier au format GraphViz, et le sommet inconnu n'est pas généré non plus. Il est donc possible que la somme des arcs entrants dans un sommet soit inférieure au nombre total de requêtes référençant l'URL cible correspondante si des sources inconnues la référencent ! Il est aussi possible que des sommets se trouvent isolés.

Utilisation de l'option -e	Ecriture dans la sortie standard : "Warning : Hits concerning images, css or javascript documents will be ignored"	**23**
Utilisation de l'option -t time	Ecriture dans la sortie standard : "Warning : Only hits between <time>h and <time+1>h will be taken into account"	**24**
La commande n'a produit aucun résultat après analyse du fichier : aucune requête ne correspond aux critères de sélection donnés.	Ecriture dans la sortie standard : "Warning : No matching hit found for this command" "Nothing to be done" Aucun fichier n'est généré. Aucune autre sortie. Terminer l'exécution avec le code 0	**25**
Utilisation de l'option -g	Ecriture dans la sortie standard : "Dot-file <gvfilename.dot> generated" Générer le fichier <gvfilename.dot>	**26**
Cas d'utilisation sans options, avec un nombre de cibles trouvées n < 10	Ecriture dans la sortie standard : "Warning : Less than 10 target documents have been found" Lister les documents par ordre décroissant en nombre de hits <URL cible> (<n> hit[s]) Terminer l'exécution avec le code 0	**27**
Cas d'utilisation sans options, avec un nombre de cibles trouvées n >= 10	Ecriture dans la sortie standard : "The 10 most popular documents are :" Lister les documents par ordre décroissant en nombre de hits <URL cible> (<n> hit[s]) Terminer l'exécution avec le code 0	**28**
Cas d'utilisation avec options	Ecriture dans la sortie standard : Lister les documents correspondant aux options par ordre décroissant en nombre de hits <URL cible> (<n> hit[s]) Terminer l'exécution avec le code 0	**29**

Des requêtes ont une URL source inconnue	Le nombre de hits de l'URL cible est incrémenté, et un sommet pour la cible est généré dans le fichier GraphViz (si on choisit l'option -g), mais l'arc correspondant à la requête n'est pas généré	**30**

II. Architecture de l'application

II.A. Rôle des classes et fonctionnement général

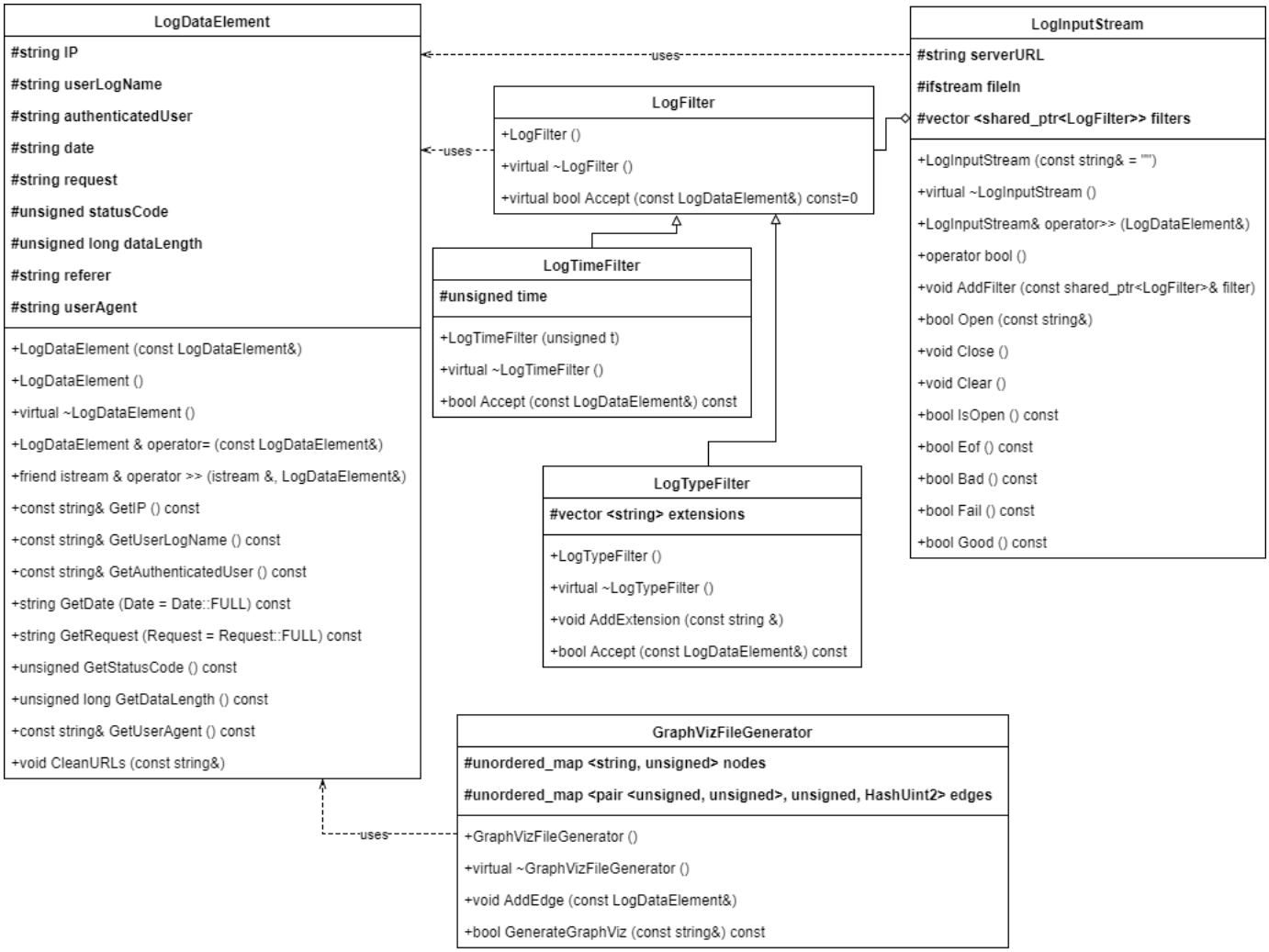


Diagramme de classes de notre application

Nous avons choisi de concevoir notre application en fonction des différents services dont nous avions besoin, que nous avons encapsulé dans des classes en veillant à ce que certaines classes clés soient réutilisables (voir section II.B). Les principaux rôles que nous avons identifiés sont les suivants :

- La lecture d'un fichier de log Apache (classe LogInputStream)
- Le stockage des informations contenues dans une entrée de log Apache, c'est à dire concernant une requête HTTP (classe LogDataElement)
- Le filtrage lors de la lecture du log, selon des critères variés (classes LogFilter, LogTimeFilter et LogTypeFilter)
- La génération d'un fichier au format GraphViz (classe GraphVizFileGenerator)

En plus de cela, le module Main gère ce qui concerne le chemin d'exécution du programme, dans la fonction main, qui s'occupe notamment de l'allocation dynamique des ressources mémoires en fonction du choix qui est fait pour le chemin d'exécution lors du passage des arguments en ligne de commande. La fonction main gère aussi l'interaction avec l'utilisateur : elle se charge de l'affichage des messages d'avertissement et d'erreur. Le module Main propose quelques fonctions utiles pour interagir avec l'environnement extérieur au programme, notamment pour vérifier l'existence d'un fichier par exemple.

La classe LogInputStream est utilisée pour extraire des objets LogDataElement à partir d'un fichier de log Apache. Elle peut pour cela utiliser une liste des filtres héritant de la classe LogFilter, qui lui permettent d'ignorer les entrées du fichier qui ne correspondent pas à des requêtes acceptées par l'ensemble des filtres.

La classe GraphVizFileGenerator est chargée de stocker les informations nécessaires à la modélisation du graphe, et propose une méthode permettant de générer ce graphe dans un fichier au format GraphViz.

Nous avons choisi de ne pas créer de classe pour stocker et trier les URL cibles par ordre de popularité, car ce service est suffisamment simple pour ne pas nécessiter de classe. Nous avons considéré qu'il était inutile de créer une classe dont le seul rôle serait d'encapsuler un conteneur de la STL pour stocker les URL cibles...

II.B. Classes réutilisables

Les classes LogInputStream, LogDataElement et les filtres héritant de LogFilter ont pour vocation d'être réutilisables. Elles peuvent s'utiliser indépendamment du programme *analog* pour extraire des données d'un fichier de log Apache de n'importe quel serveur. On peut même définir de nouvelles classes héritant de LogFilter pour avoir de nouveaux filtres selon de nouveaux critères ! De plus, LogInputStream reste fidèle à la manière dont les flux s'utilisent en C++ : elle surcharge les opérateurs >> et bool , et offre une interface similaire à ifstream : méthodes open, is_open, close, eof, fail, bad, good, clear. Un LogDataElement peut aussi être extrait à partir d'un istream, sans passer par un LogInputStream.

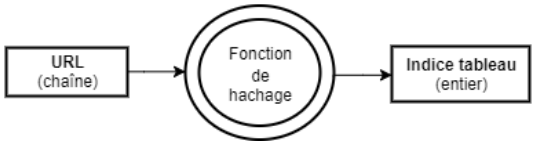
III. Structures de données

Nous utilisons deux structures de données, une pour chaque problème à résoudre : le tri des URL par popularité, et la génération du fichier GraphViz. Cela nous permet d'utiliser moins de mémoire dans le cas où on ne souhaite pas générer le fichier GraphViz : nous ne créons en mémoire que les structures nécessaires. Nous présenterons les schémas mémoire de ces structures pour le fichier de log suivant en exemple :

```
192.168.0.0 - - [08/Sep/2012:11:16:02 +0200] "GET /B.html HTTP/1.1" 200 12106 "/C.html" "Mozilla"
192.168.0.0 - - [08/Sep/2012:11:18:56 +0200] "GET /A.html HTTP/1.1" 200 11635 "/B.html" "Mozilla"
```

III.A Popularité des ressources demandées

Table de hachage : URL cibles	
URL cible (chaîne)	Hits (entier)
/B.html	1
/A.html	3



Nous utilisons une table de hachage pour stocker les URL cibles sous forme de chaînes de caractères, que nous associons avec le nombre de fois qu'elles sont atteintes dans le fichier de log (hits). La table de hachage permet d'accéder en temps constant aux éléments, si on a bien choisi la fonction de hachage, ce qui nous est fort utile car pour chaque URL extraite du fichier on veut tester si elle existe déjà dans la table. L'insertion d'une URL et la mise à jour de son nombre de hits se fera avec une excellente complexité de O(1), soit O(n) en tout pour le

fichier de log, la table de hachage est donc un très bon choix. Au moment du tri, on appliquera simplement un algorithme de Tri Rapide en O(n*log(n)). La complexité mémoire de cette table sera de O(n), où n est le nombre d'URL cibles différentes dans le fichier, à condition de bien choisir la taille de la table pour ne pas gaspiller de mémoire.

III.B. Structure pour la génération d'un fichier GraphViz

Table de hachage : Sommets	
Sommet URL (chaîne)	ID (entier)
/B.html	1
/A.html	3
/C.html	2

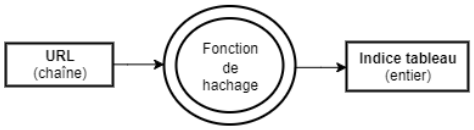
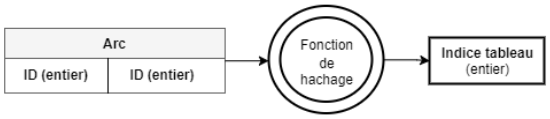


Table de hachage : Arcs		
Arc		Hits (entier)
ID (entier)	ID (entier)	
2	1	1
1	3	1



Nous n'utilisons pas une structure conventionnelle pour stocker le graphe (liste d'adjacence, matrice d'adjacence) parce que le seul traitement que nous voulons faire dessus est le stocker. La classe GraphVizFileGenerator se contente donc de stocker tous les sommets et arcs du graphe.

Nous utilisons une table de hachage pour stocker les sommets en associant à chaque URL (chaine de caractère) un identifiant unique (entier). Nous avons besoin de cet identifiant pour générer le fichier GraphViz, car une URL ne respecte pas la syntaxe prévue pour un nom de sommet. L'intérêt de la table de hachage, si on choisit bien la fonction de hachage, est d'ajouter et de rechercher un élément en temps constant O(1). Or nous avons besoin d'ajouter beaucoup de sommets et de tester aussi souvent s'ils existent déjà pour leur associer un identifiant, donc la table de hachage est une très bonne option.

Nous utilisons une table de hachage pour stocker les arcs, en associant à chaque arc (qui est une paire d'identifiants uniques entiers qui référencent deux URL, source et cible, dans la table des sommets) le nombre de fois qu'il est franchi (hits). Nous avons fait ce choix pour les mêmes raisons : nous avons souvent besoin de tester l'existence d'un arc pour incrémenter le nombre de fois qu'il est franchi, et nous n'avons pas besoin de trier les arcs. De plus, cette représentation des arcs par un couple d'identifiants entiers permet d'utiliser beaucoup moins de mémoire qu'avec un couple d'URL sous forme de chaînes de caractères.

Au final, avec n sommets différents et p arcs différents, et en choisissant une taille raisonnable pour les tables de hachage pour éviter de gaspiller de la mémoire, on a une complexité spatiale de O(n + p). L'insertion d'un nouvel arc à partir d'un LogDataElement se fait avec une complexité temporelle de O(1) en choisissant de bonnes fonctions de hachage. La génération du fichier GraphViz se fait en O(n + p) car il s'agit simplement de parcourir linéairement les deux tables de hachage.

III.C Conclusion sur les performances

L'opération qui prend en fait le plus de temps est la lecture du fichier, même si elle se fait seulement avec une complexité linéaire, car elle se fait sur la mémoire disque, qui est beaucoup plus lente que les opérations faites sur la RAM !