Compte rendu – TP C++ n°4

Analyse de logs apache

CHRISTOPHE ETIENNE | WILLIAM OCCELLI

07/02/2018

Sommaire

Introduction 2

Spécifications générales 2

Spécifications détaillées 3

Conception 5

Données 6

Schémas des structures de données 8

Conclusion et axes d'amélioration 9

Introduction

Le but de ce TP est d'analyser des fichiers de logs apache afin d'en extraire des informations. Un fichier de logs consigne toutes les opérations effectuées sur un serveur 'web dans notre cas).
Avant toute réalisation, notre démarche est structurée en 3 étapes:

- Une première étape de spécification où nous notifions de manière complète le fonctionnement du programme (cas normaux, cas limites, cas d'erreur)
- Une deuxième étape de conception où nous définissons nos classes avec pour chacune, son rôle, ses attributs et ses méthodes.
- Une troisième étape qui détaille les structures de données utilisées pour répondre au problème posé ainsi que l'évaluation des contraintes mémoire/performance associées.

Pour la réalisation, la solution est codée en langage C++.

Spécifications générales

Notion d'heure valide

Nous avons choisi de considérer les heures valides entre 00 (minuit) et 23 pour simplifier la notation PM/AM. Les minutes et secondes sont négligées lorsque nous stockons les informations d'un log car non nécessaires dans notre application.

Heure de GreenWich

Afin d'avoir une référence commune, nous avons pris la décision de stocker l'heure d'un log sur le référentiel de GreenWich. Si dans le fichier de log Apache l'heure est la suivante: 11:16:02 +0200, l'heure stockée sera 09.

Ecraser un fichier .dot existant

Lorsque le fichier .dot spécifié avec l'option -g existe déjà dans le répertoire, nous avons pris la décision d'écraser les données qu'il contient et les remplacer par les nouvelles.

Status code

Notre application retourne par défaut la liste des cibles les plus populaires (=nombre de hits). Nous avons donc pris la décision d'ignorer le status code des lignes de log. Un hit consulté est comptabilisé peu importe la valeur du status code.

Méthode http

Notre application prend en compte toutes les méthodes http.

Fichier Log

Si le fichier ne nous permet pas de faire un top 10, il y a deux possibilités. Si il ne contient aucun log, un message d'information est retourné sinon on affiche un classement avec les logs qu'il contient.

Cas d'égalité

En cas d'égalité pour le nombre de hits, nous avons décidé de garder un Top 10 et d'afficher les logs à égalité jusqu'à atteindre 10 logs par ordre d'insertion dans la structure de données.

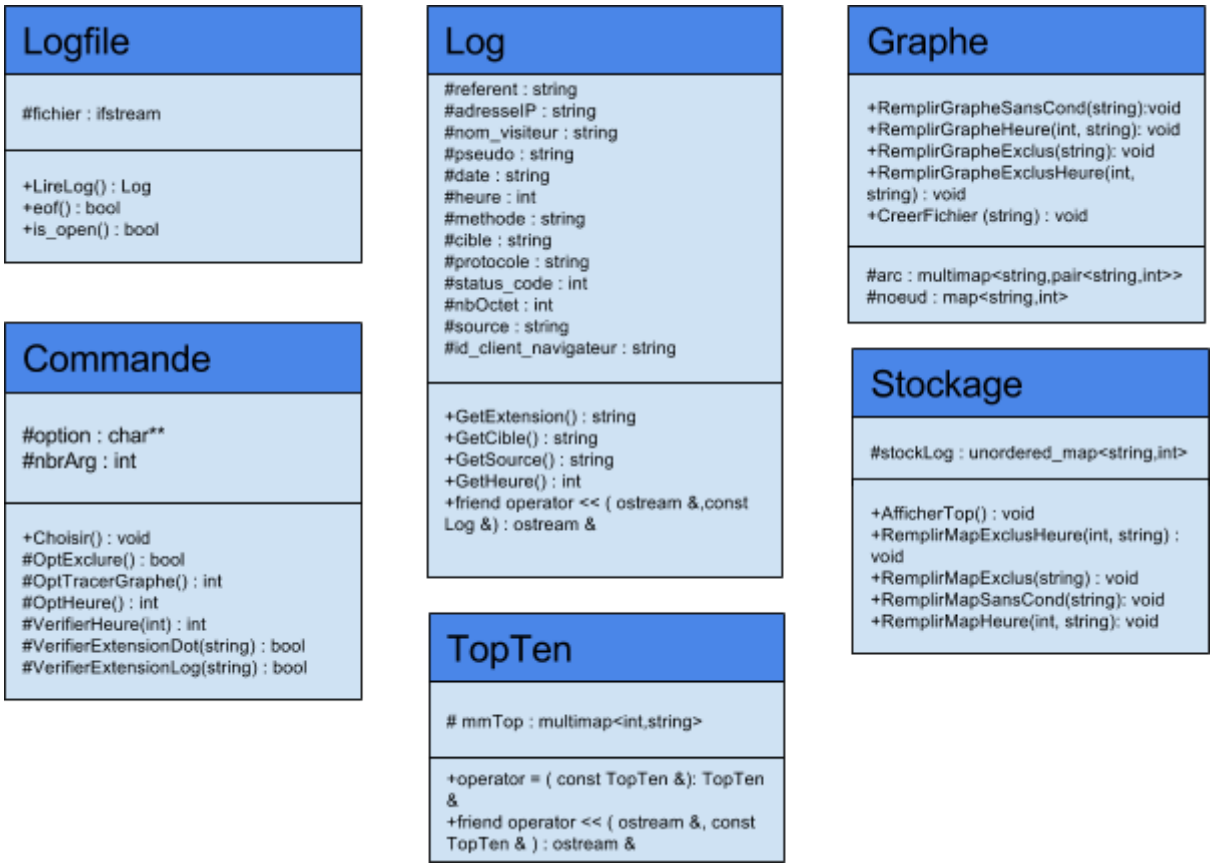
Spécifications détaillées

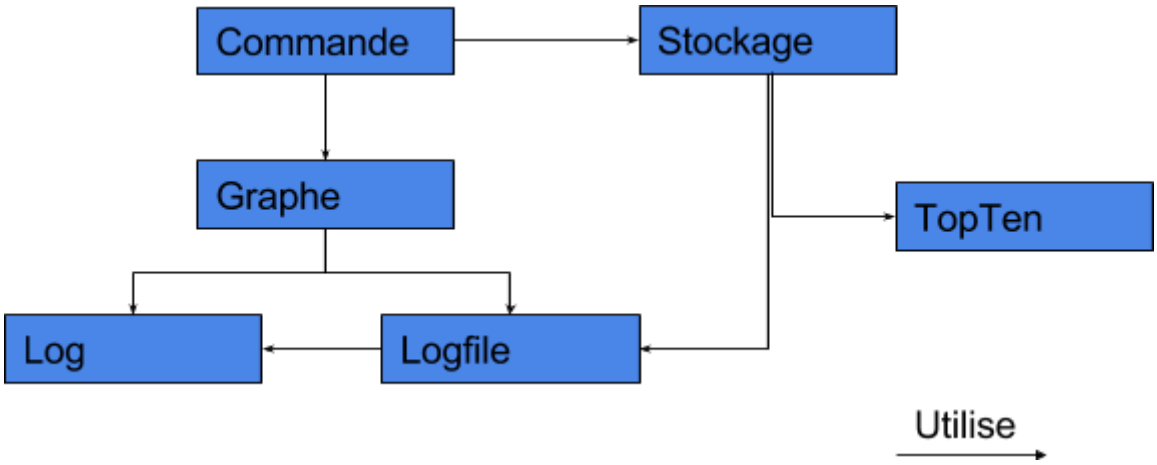
N°Test	Cas considéré	Réponse attendue
1	Aucune option et fichier log ne sont passés en paramètre.	stderr : “Problème : aucune option détectée”
2	Aucune option mais fichier log existe	stdout : “Affichage du TOP 10”
3	Extension du fichier de log non correcte	stderr : “Fichier log passé en option n'a pas la bonne extension”
4	Option -e est rentrée avec un fichier log valide	stdout : “Affichage avec les fichiers '.css', '.jpg', '.png' et '.js' exclus Affichage du TOP 10 :”
5	Fichier .log n’existe pas, aucune option	stderr : “Erreur lors de l'ouverture du fichier
6	Option -e est rentré mais l'extension du fichier log n'est pas correcte	stderr : “Fichier log passé en option n'a pas la bonne extension”
7	Option heure rentrée avec une heure valide et l'extension du fichier log est correcte	stdout : “Affichage pour un intervalle compris entre XXhXX et YYhYY Affichage du TOP 10 :”
8	Heure non comprise entre 0 et 23 avec option -t activée	stderr : “heure non valide”
9	Options -e et -t heure avec un fichier valide	stdout : “Affichage avec les fichiers '.png', '.css', '.js' et '.jpg' exclus pour un horaire compris entre XXhXX et YYhYY Affichage du TOP 10 :”
10	options -e et -t heure rentrées dans l’ordre inverse au test 9 avec un fichier log valide	stdout : “Affichage avec les fichiers '.png', '.css', '.js' et '.jpg' exclus pour un horaire compris entre XXhXX et YYhYY Affichage du TOP 10 :”
11	Options -e et -t heure rentrées avec une heure non valide et fichier log valide	stderr : “heure non valide”

12	Création d'un fichier dot à partir d'un fichier log (option -g test12.dot)	stdout : "Affichage du TOP 10 : fichier test12.dot créé"
13	Création d'un fichier dot avec comme nom de fichier .dot un fichier déjà existant	stdout : "Affichage du TOP 10 : fichier test13.dot écrasé"
14	La même option est rentrée plusieurs fois (ici -e)	stderr : "aucune option connue ou option rentrée plusieurs fois"
15	Option -t rentrée sans heure précisée après.	stderr : "option rentrée non valide"
16	Option -g est rentré sans fichier après	stderr : "option rentrée non valide"
17	Toutes les options sans fichier ni heure renseignées après leur option respectives	stderr : "Le fichier indiqué ne contient pas la bonne extension (dot)"
18	Heure composée de lettres après l'option -t avec un fichier log valide	stderr : "heure non valide"
19	Aucune option rentrée à partir d'un fichier log valide et existant mais qui ne contient aucun log	stdout : "Le fichier ne contient aucun log"
20	Création d'un fichier dot (non existant) avec l'option -e pour un fichier log valide	stdout : "Affichage avec les fichiers '.css', '.jpg', '.png' et '.js' exclus Affichage du TOP 10 : fichier test20.dot créé"
21	Le fichier cible de l'option -g n'a pas le bonne extension (.dot)	stderr : "extension du fichier n'est pas valide (ce n'est pas un fichier dot)"
22	Toutes les options avec fichier de log valide	
23	Certaines options valides, d'autres non valides	stderr : " Trop d'options rentrées"
24	Option '-t' valide rentrée plusieurs fois	stderr : "Erreur dans l'extension du fichier Dot , l'heure choisie ou les options rentrées"
25	Option '-g' est rentrée plusieurs fois de façon valide	stderr : "Erreur dans l'extension du fichier Dot , l'heure choisie ou les options rentrées"
26	Nombre valide d'options incorrectes	stderr : "option inconnue ou options rentrées plusieurs fois"

27	Options valides sans fichier log précisé	stderr : “Fichier log passé en option n'a pas la bonne extension”
28	Options rentrées dans le mauvais ordre avec des fichiers dot et log valides.	stderr : “erreur dans l'extension du fichier Dot, l'heure choisie ou les options rentrées”
29	Options -g et -t avec une heure non valide mais des fichiers dot et log valides	stderr : “Erreur dans l'extension du fichier Dot , l'heure choisie ou les options rentrées”

Conception





Données

Grâce à la conception détaillée, nous savons que les classes Stockage, Graphe et TopTen possèdent des structures de données.
Plusieurs critères ont influencé notre choix de structure:

Pour la classe Stockage:

Nous avons pensé à différentes structures de données pour stocker les logs afin d'afficher un top 10 des urls de document les plus consultées ensuite. La classe stockage n'a pas besoin de stocker les logs de manière ordonnée. En revanche, il était important de prendre en compte le nombre de fois où la cible (=url du document consulté) a été consultée depuis n'importe quel source (=url du document sur lequel se trouvait le navigateur quand il a effectué sa requête). Par conséquent, il nous fallait une structure de données capable de stocker la cible associée à son nombre de hits. Nous avons éliminé les structures séquentielles dont la complexité en recherche ou pour ajouter n'était pas optimale. Nous nous sommes penchés vers les conteneurs associatifs suivants : ABR et table de hachage. Après une évaluation des performances en fonction de nos contraintes, nous avons choisi la solution de la table de hachage. En effet, pour rechercher un élément, la complexité en temps est O(1) alors que pour un ABR la complexité en temps est O(log(n)). Comme la clé est unique, il n'y a pas de conflit (ou alors on incrémente le nombre de hits) et la performance en terme de vitesse d'exécution est meilleure. Nous avons privilégié la rapidité par rapport à la mémoire car les fichiers logs peuvent être conséquents et le traitement se faisant en local, nous n'avons pas de limitation mémoire.

Pour la classe TopTen:

Pour afficher le top 10 des cibles les plus consultées, les contraintes étaient les suivantes :

- Un top 10 nécessite un classement donc besoin d'une structure ordonnée suivant le nombre de hits
- Plusieurs fichiers peuvent avoir le même nombre de hits

- Afficher le top 10 avec le nombre de hits et la cible associée à ce nombre

Pour respecter ces contraintes, la solution présentant le meilleur ratio vitesse/mémoire était l'arbre binaire de recherche. Pour remplir cette ABR, nous parcourons une fois la première structure de données (table de hachage de stockage) et nous insérons dans l'ABR le couple clé/valeur suivant : cible/nombre de hits en inversant cette fois-ci la clé et la valeur. L'ABR présente l'avantage de trier au moment de l'insertion en fonction du nombre de hits (la clé) tout en gardant l'association avec sa cible (la valeur ici). Pour trouver les dix cibles les plus consultées, il ne nous reste plus qu'à parcourir les dix premières valeurs en partant de la droite et de les afficher dans l'ordre.
En termes de mémoire, stocker une deuxième fois les logs dans une autre structure est assez couteux mais de la même manière nous avons privilégié la rapidité.

Pour la classe Graphe:

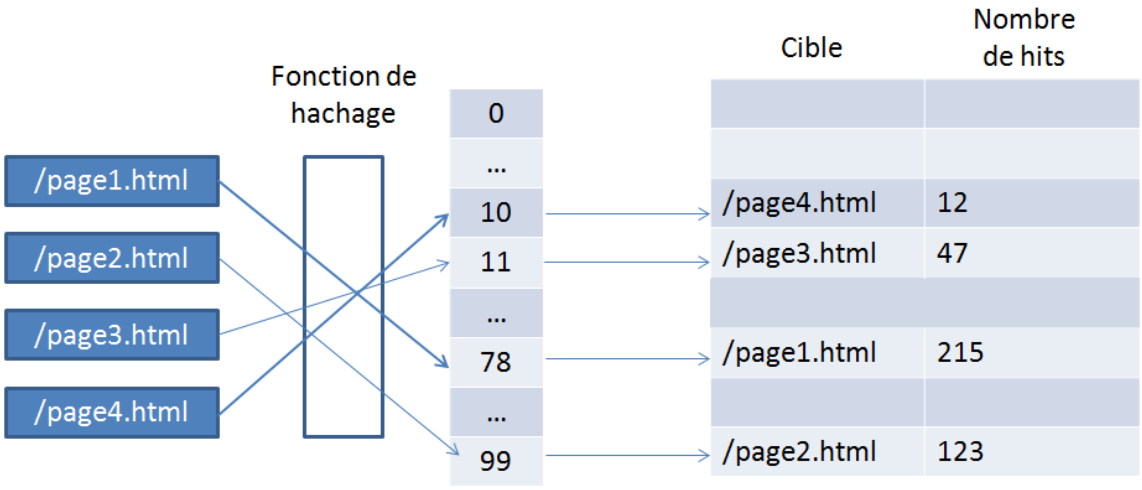
Pour cette classe, nous avons deux structures de données. Les contraintes étaient les suivantes :

- Stocker la source, la cible atteinte associée et le nombre de fois où cette cible est atteinte depuis cette source.
- Description textuelle précise pour l'outil GraphViz

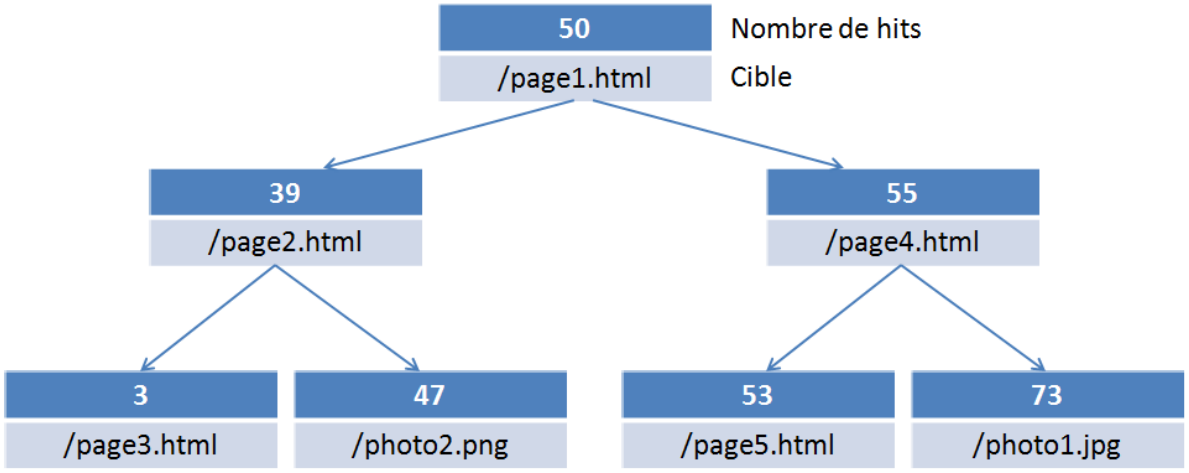
Cette contrainte princpale ainsi que le format de la description textuelle d'un graphe pour utiliser l'outil GraphViz nous a amené à choisir pour cette classe deux structures de données. En effet, en plus de stocker les informations source → cible et le nombre d'occurrences, il fallait dans la description textuelle associer un nom unique à chaque url du document consulté. Pour cela, nous avons choisi de les différencier par un identifiant distinct. Ainsi, nous avons une structure qui doit stocker la cible ou source de manière unique avec un identifiant et une structure pour stocker la source et la cible associé au nombre d'occurrences de cette même requête. Pour les deux structures, nous n'avons pas besoin que celles-ci soit ordonnées. Nous avons besoin d'y accéder rapidement en recherche. La solution que nous avons retenue est une table de hachage pour les deux. La complexité en recherche est O(1).

Schémas des structures de données

Table de hachage



Arbre Binaire de Recherche



Conclusion et axes d'amélioration

Nous avons rencontré des difficultés pendant la réalisation de ce TP.

- La première fut d'utiliser à bon escient la STL avec laquelle nous n'étions pas encore familiers. C'était la première fois que nous utilisions les structures de données et leurs méthodes associées. Nous avons essayé de les exploiter au mieux.
- De plus, nous avons rencontré des difficultés concernant les nombreux détails à prendre en compte. Lors de la première batterie de tests que nous avons voulu exécuter, nous nous sommes aperçus qu'il y avait plein de cas particuliers que nous n'avions pas pris en compte. Par exemple, lorsque l'utilisateur rentre plusieurs fois le même argument ou rentre une option '-t' valide mais dont l'heure qui suit n'est pas un nombre. Nous avons donc pu constater l'importance des tests pour perfectionner notre code et le rendre le plus sûr possible.
- Concevoir l'application en fonction des différentes structures de données auxquelles nous pouvions penser et réfléchir sur la plus appropriée pour notre application.

Concernant les axes d'amélioration envisageables, nous en avons retenu plusieurs :

- La lisibilité du code est un axe principal. Certaines méthodes pourraient être optimisées comme par exemple la méthode Choisir () de la classe Commande. Nous pourrions penser par exemple à créer une classe supplémentaire Option qui stockerait les différentes options et contiendrait plusieurs méthodes séparées gérant chaque option. Au moment de la conception, nous n'avions pas pensé qu'il faudrait prendre autant de cas particuliers en compte ce qui a alourdit le code au fur et à mesure de la réalisation des tests.
- Les messages d'erreurs ne sont pas toujours très précis. De par notre solution, il faudrait rajouter de nombreuses clauses if/else ce que nous avons décidé de ne pas faire pour ne pas que le code ne soit trop lourd.
- Il existe surement des scénarios auxquels nous n'avons pas forcément pensé et pour lesquels notre programme ne fonctionnerait pas comme il le faut.
- D'autres fonctionnalités pourraient être prises en compte par exemple la gestion des status code, des paramètres dans l'URL ou des méthodes http. On pourrait penser à interagir avec l'utilisateur pour lui demander son souhait de les prendre en compte ou non.
- Enfin, le code pourrait bénéficier d'une réelle optimisation avec une meilleure maitrise de la STL et des fonctionnalités qu'elles proposent.