Saad GUESSOUS
Jules DUCANGE

Compte Rendu de TP : C++ n°4 : Analyse de logs Apache

Binôme B3307 : Saad GUESSOUS et Jules DUCANGE

Spécifications du programme

Objectif du programme

Le programme fourni permet le traitement et l'analyse de Logs Apache, plus spécifiquement les documents auxquels ces logs font référence. L'outil permet d'élaborer des statistiques sur les différents logs, et il peut prendre en compte différents paramètres afin de générer des sorties différentes.

Spécifications générales prises en compte

Le programme a été conçu avec comme objectif de maximiser sa réutilisation et son évolutivité, pour cela :

- Le programme actuel est conçu pour traiter les logs du domaine `intranet-if.insa-lyon.fr`, toutefois, ce domaine peut être changé directement en le modifiant dans le fichier `.analog` dans le répertoire où vous exécutez le programme. Il n'y a donc pas besoin d'accéder au code pour modifier cette information.
- La classe LogReader permet de traiter toutes les informations d'un `log`. Malgré le fait que la version actuelle du programme n'est intéressée que par 3 informations (i.e le nom des documents, l'heure du log, et l'extension des documents), le programme pourrait très bien traiter les autres informations présentes dans les logs.
- Nous avons opté pour des spécifications très détaillées au niveau des paramètres du programme, cela permet d'assurer une cohérence entre ces paramètres et de faciliter l'ajout de nouveaux paramètres dans le futur.
- Lors de notre programmation, nous avons privilégié la complexité mémoire du programme. En effet, les fichiers de logs peuvent être très volumineux, et nous ne pouvons pas se permettre de dupliquer des données. (Cf partie sur le choix de la structure de données)

Autre spécification : Nous avons considéré dans notre manière de traiter les logs que le document `/exemple/` et le document `/exemple` représentent le même document, et donc n'avons pas fait de distinction entre ces deux cas.

Options de commande

Les différentes options de commande présentes dans le programme sont celles demandées dans le sujet, plus une permettant de trier les logs par validité de la requête (en utilisant le code de retour).

-g nomfichier.dot : cette option permet de produire un fichier au format GraphViz du graphe analysé.

-e : cette option permet d'exclure tous les documents ayant pour extension .css, .js ou de type image.

-t heure : cette option permet de récupérer uniquement les logs dont l'heure appartient à l'intervalle [heure, heure+1[

-i : cette option permet de récupérer uniquement les logs dont le code de retour correspond à une requête valide. C'est à dire les codes commençant par '1' (en attente) ou '2' (succès).

Liste des Tests développées

Afin de vérifier le bon fonctionnement du programme, nous avons développé un total de 17 tests permettant de tester chaque combinaison différente d'exécution du programme. La manière d'exécuter ces tests est dans le fichier readme.txt, dans le répertoire de départ.

1. Vérifie le résultat de l'exécution du programme quand il est appelé sur un fichier valide de plus de 10 documents, sans paramètre.
2. Vérifie le résultat de l'exécution du programme quand il est appelé sur un fichier valide de moins de 10 documents, sans paramètre.
3. Vérifie le résultat de l'exécution du programme quand il est appelé sur un fichier valide vide, sans paramètre.
4. Vérifie le message d'erreur quand le programme est appelé avec un fichier invalide.
5. Vérifie le résultat de l'exécution du programme quand il est appelé sur un fichier valide, avec l'option -g graph.dot
6. Vérifie le résultat de l'exécution du programme quand il est appelé sur un fichier valide, avec l'option -g sans spécifier le nom du fichier.
7. Vérifie le résultat de l'exécution du programme quand il est appelé sur un fichier valide, avec l'option -e.
8. Vérifie le résultat de l'exécution du programme quand il est appelé sur un fichier valide, sans l'option -e. (Vérifie l'apparition de fichier .css)
9. Vérifie le résultat de l'exécution du programme quand il est appelé sur un fichier valide sans document à ignorer, avec l'option -e.
10. Vérifie le résultat de l'exécution du programme quand il est appelé sur un fichier valide composé uniquement de document à ignorer, avec l'option -e.
11. Vérifie le résultat de l'exécution du programme quand il est appelé sur un fichier valide, avec l'option -t 10.
12. Vérifie le résultat de l'exécution du programme quand il est appelé sur un fichier valide, sans l'option -t 10. (Vérifie l'apparition de nouveau log)
13. Vérifie le résultat de l'exécution du programme quand il est appelé sur un fichier valide composé de requêtes effectuée entre 10 et 11h, avec l'option -t 10.
14. Vérifie le résultat de l'exécution du programme quand il est appelé sur un fichier valide composé de requêtes effectuée entre 16 et 17h, avec l'option -t 10. (Aucun résultat)
15. Vérifie le résultat de l'exécution du programme quand il est appelé sur un fichier valide, avec l'option -t sans préciser d'heure.
16. Vérifie le résultat de l'exécution du programme quand il est appelé sur un fichier valide, avec l'option -t avec une heure invalide : quinze-heure.
17. Vérifie le résultat de l'exécution du programme quand il est appelé sur un fichier valide, avec toute les options.

Choix de la structure de données

L'objectif principal de nos choix était de ne pas dupliquer les noms des documents (i.e les string) et de les stocker en mémoire une seule et unique fois. Pour cela, nous avons opté pour les structures de données suivantes :

- `unordered_map<string, size_t> store` : permet d'attribuer à chaque nom de document un numéro (i.e clé) unique qui lui est associé
- `unordered_map<size_t, unordered_map<size_t, size_t>> Graph` : cette structure permet de stocker, pour chaque nom de document équivalent à un `target`, tous les documents ciblant cette target, avec pour chacun le nombre de hits.
- `map<size_t, size_t> Hits` : permet de stocker pour chaque nom de document, le nombre de hits total ciblant le document.

Complexité mémoire

Si on ne considère que les objets de type `string` (qui prennent beaucoup plus de mémoire que les `sizes_t`), alors la complexité mémoire du programme est en O(n), avec n le nombre de documents uniques dans le fichier de logs.