Analog Compte rendu

Karim Benhmida et Mehdi Kitane (B3409)

Introduction

Analog est un programme qui s'exécute en ligne de commande, qui permet d'analyser des fichiers de log du serveur HTTP Apache. Ce programme est capable de générer des statistiques sur l'ensemble du fichier log, mais aussi en spécifiant certains critères comme l'heure de visite et le type de document visité. Par ailleurs, il peut aussi générer un document synthétique au format GraphViz (qui pourra ensuite être facilement traduit en graphe).

Généralités

1 Définitions

Apache : Serveur HTTP.

Log : Fichier qui consigne un certain nombre d'informations (adresse IP, document demandé, heure, code retour…) pour chaque requête reçue sur le serveur.

2 Choix généraux

Option -l : Cette option, qui limite les noeuds du graphe à prendre en compte, ne pourra être utilisée qu'en présence de l'option qui génère le graphe (option -g).

Passage de paramètres : Chaque URL sera considérée comme une page unique, même si certains paramètres sont passés dans celle ci (page.php?articleid=34 n'est pas la même page que page.php). L'influence des paramètres sur une page varie d'un site à l'autre, pour certains sites elle permet d'ajouter des informations sur le visiteur sans changer la page, mais dans d'autres cas, la page change complètement avec les paramètres.

Informations extraites : Dans un soucis de réutilisabilité, le programme va extraire depuis le fichier log les données suivantes : Adresse IP, date, heure, type de requête, cible, referer et code retour. Nous n'utiliserons pour ce TP que l'heure, la cible et le referer.

Spécifications du programme

Lancement sans options

$./analog nomfichier.log (Test 18ex)

Affichera sur la console la liste des 10 documents les plus consultés par ordre décroissant de popularité.

Lancement avec une option

$./analog -g nomfichier.dot nomfichier.log (Tests 15g, 16g, 17g, 26)

- Génère un fichier nomfichier.dot (dans le même répértoire) en considérant toutes les entrées du log.
- Affiche un top 10 des documents les plus consultés par ordre décroissant de popularité.

$./analog -l nbhits nomfichier.log (Tests 1l, 2l, 3l, 4l, 5l)

- l'option -l est ignorée, car -g n'est pas active.
- Affiche un top 10 (comme indiqué plus haut).

$./analog -x nomfichier.log (Test 6x)

- Affiche un top 10 en ignorant certains types de fichier (images, css, js).

$./analog -t heure nbhits nomfichier.log (Tests 7t, 8t, 9t, 10t, 11t, 12t, 13t, 14t)

- Affiche un top 10 en ne considérant que les requêtes effectuées entre heure et heure+1.

Lancement avec 2 options

	-g	-l	-x	-t
-g	Seule la dernière option est prise en compte.Revient à exécuter le programme avec -g	Génère fichier .dot avec un nombre de hits des noeuds supérieur à nbhits de -l.	Génère fichier .dot en excluant les images, css et js. (Test 19ex)	Génère fichier .dot en ne considérant que les requêtes entre heure et heure+1. (Test 20ex)
-l		Dernière option prise en compte. Revient à exécuter le programme avec -l	Option -l ignorée (car exécutée sans -g). Revient à activer -x seule.	Option -l ignorée. Revient à activer -t seule.
-x			Pareil que -x seule.	Affiche Top 10 des documents consultés entre heure et heure +1, en excluant les images, css et js.
-t				Seule la dernière option est prise en compte.

Lancement avec 3 options

$./analog -g nomfichier.dot -l nbhits -x nomfichier.log

- Génère un fichier nomfichier.dot en ne considérant que les entrées du log avec un nombre de noeuds qui sont liés par un arc avec un nombre de hits supérieur à *nbhits* et en excluant les documents de type image, css et js.
- Affiche un top 10 des documents les plus consultés du log en excluant les documents de type image, css et js.

$./analog -g nomfichier.dot -l nbhits -t heure nomfichier.log

- Génère un fichier nomfichier.dot en ne considérant que les entrées du log comprises entre heure et heure+1 et avec un nombre de noeuds qui sont liés par un arc avec un nombre de hits supérieur à *nbhits*
- Affiche un top 10 des documents les plus consultés du log entre heure et heure+1.

$./analog -g nomfichier.dot -t heure -x nomfichier.log (Test 21ex)

- Génère un fichier nomfichier.dot en ne considérant que les entrées du log comprises entre heure et heure+1 et qui ne sont pas des images, css et js.
- Affiche un top 10 des documents les plus consultés du log entre heure et heure+1 en excluant les images, css et js.

$./analog -l nbhits -t heure -x nomfichier.log

- l'option -l est ignorée.
- Affiche un top 10 des documents les plus consultés du log entre heure et heure+1 en excluant les images, css et js.

Lancement avec 4 options

$./analog -g nomfichier.dot -l nbhits -x -t heure nomfichier.log (Test 30)

- Génère un fichier nomfichier.dot en ne considérant que les entrées du log comprises entre heure et heure+1, qui sont liés par un arc avec un nombre de hits supérieur à *nbhits* et qui ne sont pas des images, css et js.
- Affiche un top 10 des documents les plus consultés du log entre heure et heure+1 en excluant les images, css et js.

Tests fonctionnels

L'ensemble des tests fonctionnels réalisés avec leur description se situe dans le dossier /Tests à la racine du projet.

Architecture de l'application

Nous avons divisé notre application suivant ce schéma afin de privilégier la réutilisabilité :

- Un **namespace** commande qui contient les fonctions qui aident à analyser une commande :
    - Vérification de la syntaxe des fichiers
    - Détection des options activées
    - Vérifie si une option a besoin d'un paramètre
    - Vérifie si le paramètre est valide

- Une **classe** Log contient toutes les méthodes nécessaires à la manipulation d'un fichier log :
    - lecture d'un fichier
    - analyse par ligne
    - Recherche de séparateur dans une ligne
    - insertion des données dans une structure
    - affichage des 10 documents les plus visités
    - Détection des documents images, css et js

- Une **classe** Graph contient les méthodes utilisées pour générer un fichier au format GraphViz :
    - Générer un graphe à partir d'un objet Log.
    - Écriture d'un graphe dans un fichier .dot

Le fichier main.cpp contient la méthode main du programme qui en fonction de la commande et des options va choisir les méthodes à exécuter.

Spécification des méthodes

Classe Log

Analyse de ligne : analyseLigne

Algorithme :

```
boolean analyseLigne (chaine de caractere ligne, ligneLog *l) {

Chaine urlLocale "http://intranet-if.insa-lyon.fr"

 adresseIp = On recupère du Debut de la ligne jusqu'au premier espace
 int i = Position de la premiere occurence du caractere ':'
 heure = sousstring(i+1 jusqua i+2);  // On recupere les deux prochains caracteres

date = Position du caractère [ jusqu'au caractère ]
typeAction = Prochain caractère " \" " jusqu'au prochain espace
cible = Position fin de typeAction jusqu'a la position du prochain espace

  curseur = On se positionne au prochain guillemet " \" "
  returnCode = Position curseur jusqu'a la position du prochain espace

  referer = Position du prochain guillemet " \" " jusqu'a la position des prochains guillemets " \" "

 Si referer contient la urlLocale
          referer= referer-urlLocale
  FinSI
}
```

Cette méthode agit comme un parser, elle va parcourir la chaine de caractères passée en paramètre caractère par caractère, pour tenter d'identifier la cible, le referer et l'heure d'accès.

Une fois qu'elle aura réussi a récupérer ces éléments, elle renseignera les pointeurs correspondants en paramètre.

10 documents les plus vus : afficherDix

Affiche sur la sortie standard les 10 (au maximum, dépend du nombre d'entrées du fichier log) documents les plus visités avec le nombre de visites à côté de chaque document.

Algorithme :

```
sans retour afficherDix() {
        tableau[ tableau[2] ] top10

        tant que(il reste des cibles dans structure)
        nombreDeHits = 0 // nombre de hits pour la cible en cours
        tant que (il reste des referers pour cette cible)
                Si(option t active)
        nombreDeHits = Structure[cibleActuelle][refererActuel].tableauHeures[heureOptionT]
        Sinon
        Tant que (il reste des cases dans le tableau d'heures)
                nombreDeHits = Structure[cibleActuelle][refererActuel].tableauHeures[i]
        Fin tant que
        Fin Si

        Si(dernier referer de la cible en cours)
        Si(cible est parmis les 10 premieres de la structure)
                        AjouterATop10(indexCible, nombreDeHits) // Ajoute a la fin
                        OrganiserParOrdreDESC(top10)
        Sinon si (nombreDeHits > top10.dernierElement)
                        SupprimerElement(top10)
                        AjouterATop10(indexCible, nombreDeHits)
                        OrganiserParOrdreDESC(top10)
        Fin Si
              Fin Si
           Fin Tant que
         Fin tant que

        Parcourir top10
        afficher : Index[idCible en cours] (nombre de hits)
}
```

Lecture d'un fichier .log : lire

À partir de l'adresse d'un fichier .log passée en paramètre comme chaine de caractère, cette méthode va tenter de lire le contenu de ce fichier ligne par ligne. Pour chaque ligne, elle fait appel à la méthode analyseLigne qui va respectivement insérer la cible, referer et heure dans des variables. Elle va ensuite, en fonction de l'état des options, faire appel à la méthode de remplissage de la structure.

Recherche de caractère : chercherChar

Cette méthode recherche un séparateur donné dans une chaine de caractères à partir d'une position. Elle prend en paramètre la chaine de caractères dans laquelle rechercher, sa taille, la position de début et le caractère séparateur recherché. Si elle trouve le séparateur, elle renvoie sa position, sinon elle renvoie -1.

Insertion dans la structure de données : remplir

À partir des paramètres cible, referer et heure, cette méthode va tenter de remplir la structure de données et l'index.

Detection des images, css et js : isAsset

Détecte si une chaine de caractère contient certains mots prédéfinis. Si elle en retrouve un, elle renvoie vrai, sinon elle renvoie faux.

Classe Graph

Génération du graphe : genereGraphViz

Algorithme avec option -l :

```
Debut Methode
 string phrase = "digraph{\n"
 string noeuds = string arcs = ""
  set noeudsAjoutes

 Pour tous les elements de notre map de cible
    Pour tous les elements de notre map de referer
       entier nbClic = On calcule le nombre de clic total pour le couple cible/referer donne
       Si nbClic > (nombredehits specifie dans la commande)
          On verifie si la structure noeudAjoute contient la cible
          Si Oui
             Ne rien faire
          Si non
             Ajouter a la structure noeudAjoute la cible
             noeud += "node (numeroCibleDansLindex) [label=\ nomCible];
           FinSi

          On verifie si la structure noeudAjoute contient le referer
          Si Oui
             Ne rien faire
          Si non
             Ajouter a la structure noeudAjoute le referer
             noeud += "node (numeroRefererDansLindex) [label=\ nomReferer];
          FinSi

          arcs += "node (numeroRefererDansLindex) -> node (numeroCibleDansLindex) [label="(nbClic)"];
       Sinon
          On ne fait rien
     FinSi
    Fin Pour
 Fin Pour

 phrase = phrase + noeuds + arcs + "}"
 Retourner phrase
Fin Methode
```

Cette méthode prend en paramètre une map de cible et de référer et leur index correspondant, elle nous génère alors la structure GraphViz correspondante à cette map et cet index. Par soucis d'optimisation, deux algorithmes ont étés codés, le premier correspondant à l'option l activée et le second correspondant à la génération du graphe sans l'option l activée.

Ecriture dans le fichier .dot : ecrireGraph

À partir de l'attribut de l'objet de la classe Graph (string nomGraph), cette méthode génère un fichier .dot qui contient le résultat de la méthode genereGraph.

Namespace commande

Vérification du nom de fichier Log : checkValidNameFile

Vérifie que le nom de fichier inséré à la fin de la commande correspond bien à un fichier .log ou .txt. Retourne vrai si fichier valide.

Vérification du nom de fichier GraphViz : checkValidDotFile

Vérifie que le paramètre de l'option -g correspond bien à un fichier .dot. Retourne vrai si fichier valide.

Détection d'options : checkIfOption

Vérifie si une chaine de caractère passée en paramètre correspond à une option et récupère le paramètre pour certaines options. Retourne vrai si la chaine de caractères est une option.

Vérifie si besoin d'un paramètre : checkIfNeedParam

Vérifie si une option a besoin d'un paramètre. Si oui, retourne vrai.

Vérifie si un paramètre est valide : checkIfValidParam

Vérifie si la syntaxe d'un paramètre est valide, si oui retourne vrai.

Nom d'option : getNameOfOption

Retourne le nom de l'option sans le tiret qui la précède.

Structure de données

Les informations nécessaires (cible, referer et heure d'accès) seront insérées dans la structure représentée ci dessous. Pour éviter la redondance des adresses Web et réduire l'utilisation mémoire du programme, nous allons utiliser un index pour les cibles et les referer.
Nous avons décidé d'utiliser un arbre pour rendre la recherche des données plus rapide (complexité en log(n)).

<table>
<tr><th colspan="2">Structure (Arbre)</th></tr>
<tr><th>Clé Index Cible</th><th>Arbre</th></tr>
<tr><td>1</td><td>
<table>
<tr><th colspan="2">Arbre</th></tr>
<tr><th>Clé Index referer</th><th>Tableau [24]</th></tr>
<tr><td rowspan="3">2</td><td>2</td></tr>
<tr><td>…</td></tr>
<tr><td>34</td></tr>
<tr><td rowspan="3">4</td><td>0</td></tr>
<tr><td>…</td></tr>
<tr><td>21</td></tr>
<tr><td>…</td><td></td></tr>
</table>
</td></tr>
<tr><td>2</td><td></td></tr>
<tr><td>…</td><td></td></tr>
</table>

Après avoir décidé d'utiliser cette structure, nous voulions être surs d'avoir fait un choix cohérent, c'est pour cela que nous avons fait un comparatif entre une celle ci et une structure semblable, mais sans index (donc redondante). Voici les résultats de l'analyse du fichier anonyme.log sans options :

	Structure avec index (**choisie**)	Structure sans index avec redondance
Mémoire utilisée (**Mo**)	2,6	15,2
Temps d'exécution (**s**)	0,95	0,47

Tests effectués sur un ordinateur avec processeur 4 coeurs Intel Core i7 2,3 GHz Haswell.

On remarque que le programme avec structure sans index a besoin de 5 fois plus de mémoire pour un gain de performance qui n'est pas très élevé (0,5 secondes).
La structure avec index nous permettra donc d'analyser des fichiers logs bien plus grands avec un minimum de mémoire utilisé (Il faut savoir que l'évolution de l'utilisation de la mémoire en fonction de la taille du log n'est pas linéaire, car les mêmes pages vont se répéter plusieurs fois).