FALLOUH Marc
VIRSOLVY Mathieu

SPECIFICATIONS

1. Spécifications générales du programme.

Préambule : pour répondre au sujet nous avons opté pour une solution visant la réutilisabilité. Tous les traitements qui auraient pu être effectués à la lecture sont repoussés au dernier moment. Les données du fichier sont stockées dans leur totalité et ne sont jamais altérées. Ainsi, il est possible de réaliser « à volonté » les traitements et filtrages spécifiés en option de ligne de commande, comme s'il s'agissait de méthodes appelées depuis une interface homme machine, par exemple. Ce choix de conception induit des traitements lourds.
Pour autant, nous avons tenté d'optimiser au mieux le stockage des données pour répondre efficacement aux attentes du sujet.

Fonctionnement du programme : le programme s'exécute en ligne de commande da la manière suivante :

```
./TP3 [-e] [-t heure] [-g nomFichierSortieGraphe.dot] fichierDeLogs.log
```

Sans options, le programme renvoie la liste des 10 documents les plus consultés (on ne considère que les requêtes GET), classés par nombre décroissant de consultation. En cas d'égalité qui déborderait du classement, toutes les adresses impliquées dans l'égalité sont affichées (test unitaire : testTop10CollectionEgalite).
L'ordre des options n'a pas d'importance :
L'option e permet d'exclure les documents de type image, javascript et feuille de style CSS.
L'option t permet de sélectionner les requêtes qui se sont produites dans l'intervalle de temps [heure, heure+1]. L'heure prise en compte est l'heure de Greenwich.
L'option g permet de générer un fichier au format GraphViz. Ce fichier permet de créer un graphe montrant les interactions entre documents.

Des tests d'intégrations sont réalisés à partir du Framework de test, pour chaque exécution possible, et pour chaque cas de fonctionnement (normale, limite : fichier de log vide, erreur : logs mal formés). Ces tests sont effectués afin de s'assurer de la synergie entre classes, les autres cas particuliers de fonctionnements sont spécifiques à chaque classe (du fait de l'encapsulation des données), et testés de manière unitaire.

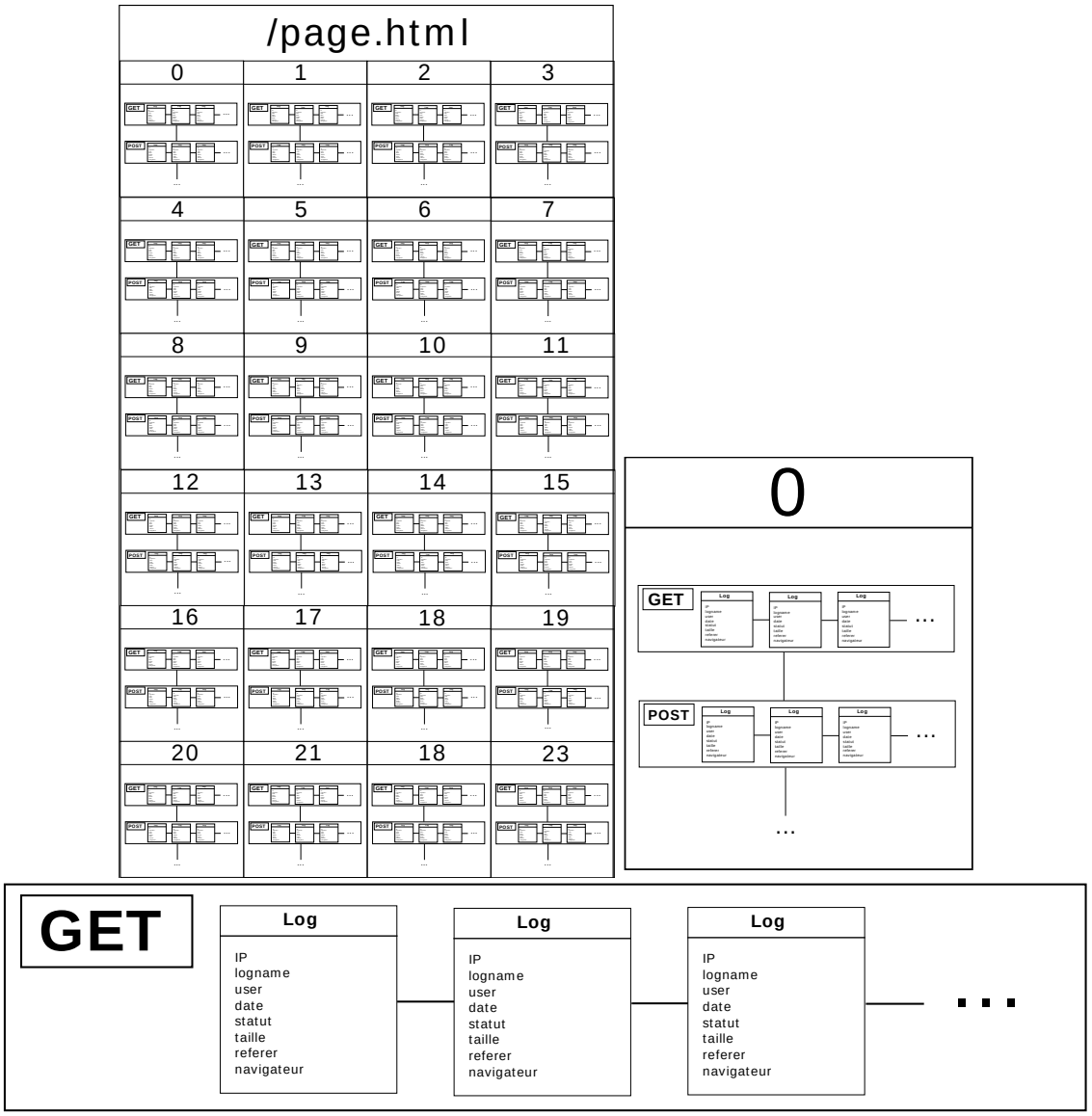
Options	Normal	Limite	Erreur
Aucune	Test1 (no opt)	Test1 (no opt) limite	Test1 (no opt) err
e	Test2 –e	Test2 –e limite	Test2 –e err
t	Test3 –t	Test3 –t limite	Test3 –t err
e+t	Test4 –e –t	Test4 –e –t limite	Test4 –e –t err
g	Test5 –g	Test5 –g limite	Test5 –g err
g+e	Test6 –e – g	Test6 –e – g limite	Test6 –e – g err
g+t	Test7 –t –g	Test7 –t –g limite	Test7 –t –g err
g+e+t	Test8 –e –t –g	Test8 –e –t –g limite	Test8 –e –t –g err

Nom du test d'intégration en fonction des options et du cas fourni par le fichier d'entrée

2. SCHEMA DES DONNÉES

[COLLECTION]

[CIBLE]



Les données sont organisées de la façon suivante :
Les informations contenues dans une ligne de log sont stockées dans la classe Log, sous forme d'attributs. Toutes les informations sont présentes à l'exception du type de requête et de l'adresse de la cible car ces informations sont présentes dans les classes contenant les Log.
Les Log sont stockés dans la classe Cible. Comme son nom l'indique, elle représente une adresse cible d'une requête. Cette classe contient des arbres de listes de Log, identifiées par le type de requête (clé de nœud). Ces arbres sont eux-mêmes stockés dans un tableau de taille 24. Ainsi, si l'on recherche les logs entrés à une certaine heure, correspondant à des requêtes GET, nous n'avons pas besoin de parcourir l'ensemble des logs. Il suffit de chercher la bonne case, puis la bonne requête, pour trouver la liste de Log. (Remarque : ce schéma de donnée relativement complexe est adapté pour faciliter les recherches spécifiques à l'énoncé. Une telle encapsulation des données présente un intérêt particulier dans le cas de fichiers de logs qui comportent beaucoup de données.)
L'ensemble des Cible se trouve dans une Collection, correspondant à un fichier de logs apache. Cette classe contient un arbre de Cible, identifiées par leur adresse. Ce regroupement facilite les recherches et la création de graphe : les logs sont rangés par adresse cible, heure et type de requête.

3. CLASSES

LOG :

Il s'agit de la classe stockant l'équivalent d'un log. On y retrouve toutes les informations excepté le type de requête (GET, POST…) et l'adresse cible (pour éviter des redondances). Toutes les données sont des chaines de caractères (string).

Attributs :

- ip : l'adresse IP à l'origine de la requête
- logname : le nom d'utilisateur du visiteur
- user : le nom d'utilisateur authentifié
- date : la date de la requête, ainsi que le décalage par rapport à Greenwich
- statut : le retour de la requête HTML
- taille : quantité de données transférées en octets
- url : l'adresse complète du referer
- referer : adresse depuis laquelle a été émise la requête
- navigateur : infos navigateur

Méthodes publiques :

- Constructeur (string &log) : découpe la chaine de caractères correspondant à un log et initialise les attributs. (testLogConstruct) Vérifie si le log est du bon format, si non, renvoie un message d'erreur (sortie erreur). (testLogConstruct2)
  Tracé si option de compilation MAP.
- Constructeur de copie : copie des attributs.
  Tracé si option de compilation MAP.
- Destructeur() : n'effectue aucun traitement, car aucune allocation dynamique
  Tracé si option de compilation MAP.

CIBLE :

Il s'agit de la classe stockant l'ensemble des logs relatifs à une adresse cible. Les données sont organisées par heure de Greenwich et type de requête, pour accélérer les traitements.

Attributs :

- lesLogs : stock les logs relatifs à une adresse cible. C'est un tableau de 24 cases (1 par heure) contenant un dictionnaire associant à un type de requête, une liste de Logs.

Méthodes publiques :

- Constructeur () : initialise le tableau lesLogs de manière dynamique. (testCibleConstruct)
  Tracé si option de compilation MAP.
- Constructeur de copie : comportement de copie par défaut (copie de lesLogs)
  Tracé si option de compilation MAP.
- Destructeur () : n'effectue aucun traitement, car aucune allocation dynamique
  Tracé si option de compilation MAP.
- int/void Ajouter(string & log) : Crée un Log à partir de la chaine de caractères entrée en paramètre. Ajoute ce Log au bon emplacement (bonne heure, bonne requête) (testCibleAjouter). Crée au besoin une nouvelle clé (requête) dans la map (testCibleAjouter2).
- int Compte(const string &requete, const bool e =false , const int t =-1) : compte le nombre de logs consignés correspondant à la requête entrée en paramètre (testCibleCompte). Tient compte des options e et t (testCibleCompteE, testCibleCompteT, testCibleCompteET) pour filtrer les résultats.

COLLECTION :

Cette classe représente une collection de logs, ordonnée pour faciliter les traitements. On la construit à partir d'un fichier de logs au format .log. Comme mentionné sur le schéma, cette classe est un dictionnaire de Cible identifiées par leur adresse.

Attributs :

- pages : un dictionnaire de Cible, qui associe l'adresse d'une page (une chaine de caractère) à l'élément de la classe Cible qui lui correspond.

Méthodes publiques :

- Constructeur(const string &nomFichier) : crée une Collection à partir d'un nom de fichier. Si le fichier n'est pas trouvé, affiche un message d'erreur. (testCollectionConstruct)
  Tracé si option de compilation MAP.
- void Top10(const bool e = false, const int t = -1) : affiche dans la console les 10 documents les plus consultés (testCollectionTop10). Si la collection comporte moins de 10 cibles, n'affiche que les cibles présentes dans la collection (testCollectionTop10Moins10). Prend en compte les options e et t (testCollectionTop10E, testCollectionTop10T, testCollectionTop10ET) pour filtrer l'affichage.
  Si jamais il y des égalités qui ne sortiraient pas si l'on n'affichait que 10 résultats, on affiche exceptionnellement davantage de résultats (jusqu'à qu'il n'y ait plus d'égalité) (testCollectionTop10Egalite).
- Destructeur : n'effectue aucune opération, pas d'allocation dynamique.
  Tracé si option de compilation MAP.

GRAPHE :

Cette classe représente un graphe. Elle contient les informations nécessaires à sa génération (nœuds et liens). On construit un Graphe à partir d'une Collection et des options souhaitées.

Attributs :

- nœuds : dictionnaire de nœuds associant à chaque adresse de page son numero pour le tracé du graphe.
- liens : dictionnaire de liens, associant à chaque paire (structure : referer et cible, identifiés par leurs numéros de noeud) son nombre de hits.

Méthodes publiques :

- void GenereFichier (const string &nomFichier) : écrit dans (ou crée) le fichier de nom « nomFichier » les instructions de création de graphe, au format spécifié dans l'énoncé. (testGrapheGenereFichier)
- Constructeur(const Collection &aCol, const bool e = false, const int t =-1) : parcourt la Collection entrée en paramètre afin de remplir les dictionnaires nœuds et liens (testGrapheConstruct). Prend en compte les options e et t (testGrapheConstructE, testGrapheConstructT, testGrapheConstructET).
  Tracé si option de compilation MAP.
- Destructeur() : n'effectue aucun traitement, pas d'allocation dynamique.
  Tracé si option de compilation MAP.