Rapport TP4 C++

Horia Burca et Ziggy Vergne, binôme B3407

January 31, 2017

1 Introduction

Dans le cadre de ce TP de C++, nous avons utilisé nos connaissances en matière de programmation orientée objet, de gestion de flux et de la STL pour concevoir une application qui puisse analyser des logs Apache et produire des fichiers de statistiques résumant l'activité web des visiteurs.

2 Spécifications générales de l'application

L'application devra être conforme aux spécifications suivantes:

- Lire un fichier texte de log Apache formaté donné en paramètre afin de pouvoir en extraire les dix ressources pour lesquelles le nombre de requêtes GET est le plus élevé. Pour cela, l'utilisateur devra spécifier le nom du fichier qu'il souhaite lire en dernier paramètre de la commande: `./analog fichierLog.log`
- Générer un fichier texte formaté pour être lu par l'outil GraphViz représentant un graphe du parcours des utilisateurs sur le site. Ce fichier portera un nom donné en paramètre du programme. Pour cela, l'utilisateur devra utiliser l'option `-g` suivie du nom du fichier de sortie :

  ```
  ./analog -g fichierDeSortie.dot fichierLog.log
  ```

- Et pour ces fonctionnalités, on donnera la possibilité de sélectionner certaines requêtes du fichier source selon les caractéristiques suivantes :
  - La plage horaire à laquelle les requêtes appartiennent avec l'option `t` suivie de l'heure de départ, l'heure de fin étant l'heure suivante:`./analog -t 10 fichierLog.log`
  - L'extension de la ressource cible. Ainsi, on donne à l'utilisateur l'option de ne pas sélectionner les requêtes dont la cible est une image, une feuille de style CSS ou un script javascript avec l'option `e` :`./analog -e fichierLog.log`
  - Une combinaison des deux options précédentes

3 Spécifications détaillées et tests associés

Après avoir défini de manière générale le comportement de notre application, nous passons à la rédaction des spécifications détaillées. Ces spécifications détaillées nous permettent de décrire avec précision le comportement que notre application devra adopter selon les circonstances.

Cas	**Comportement**	**Tests associés**
Le fichier de log n'est pas fourni en paramètre	Le programme finit par une erreur : fichier d'entrée manquant	Test01
Le fichier de log n'existe pas.	Le programme finit par une erreur : fichier illisible	Test02
Le fichier de log est protégé en lecture	Le programme finit par une erreur : fichier illisible	Test03
Le fichier de log est vide	Le programme finit par une erreur : fichier vide	Test04

Le fichier de log est protégé en écriture	Le programme s'exécute normalement	Test05
Le programme est lancé avec des paramètres inconnus	Le programme finit par une erreur : paramètre inconnu	Test06
On répète un paramètre plusieurs fois	Le programme tient compte que de la dernière répétition	Test07,Test24
Dans le fichier de log, une ressource "s'auto-référence"	Le programme s'exécute sans erreurs en tenant compte de cette requête	Test08
Le paramètre -e est spécifié	Les requêtes portant sur les fichier aux extension "ICO", "JPG", "PNG", "BMP","GIF","TIF","CSS","JS", "ICS" sont ignorés, peu importe la casse	Test09
Il y a des ressources ou référenceurs de location identique mais qui contiennent des paramètres de chemin(';') ou de requête('?') différents	La partie qui contient les paramètres de chemin ou de requête d'une ressource ou d'un référenceur est ignorée	Test10
Le paramètre `-t` est spécifié et l'heure h suivante est comprise entre 0 et 24	Les requêtes dont l'heure est comprise dans l'intervalle [h,h+1[sont sélectionnées et le programme affiche un avertissement sur la sortie standard pour notifier l'utilisateur de la restriction horaire appliquée.	Test11, Test15, Test16, Test21
Le paramètre `-t` est spécifié mais l'heure est manquante ou elle n'est pas au bon format (nombre entre 0 et 24)	Le programme finit par une erreur : l'heure n'a pas été indiqué ou l'heure n'est pas de bon format	Test12-13
Le paramètre `-t` est spécifié et l'heure indiqué est 24	Le programme s'exécute avec la restriction horaire [0, 1[	Test14
Le programme est lancé avec deux paramètres de restrictions en même temps : `-t heure -e` ou `-e -t heure`	Le programme applique les deux restrictions sur les requêtes	Test17
Il y a plusieurs ressources avec le même nombre de hits	Les ressources avec le même nombre de hits sont triées alphabétiquement	Test22
Il y a plus de 10 ressources cibles différentes référencées dans le fichier de log.	Le programme affiche que les 10 premières ressources	Test22
Le paramètre `-g` est spécifié sans nom de fichier en sortie	Le programme finit par une erreur : fichier sortie manquant	Test23
Le paramètre `-g` est spécifié et le fichier indiqué est non-vide	Le programme finit par une erreur : fichier sortie non-vide	Test18
Le paramètre `-g` est spécifié et le fichier indiqué est protégé en écriture	Le programme finit par une erreur : fichier sortie illisible	Test19
Dans une requête, le référenceur commence par "http://intranet-if.insa-lyon.fr"	Le programme enlève la racine de l'URI	Test20
Dans une requête, le référenceur ne commence pas par "http://intranet-if.insa-lyon.fr"	Le programme garde que la racine en enlevant la partie "http://" et la partie après le premier "/"	Test20
On définit une restriction horaire, sur les extensions et on donne un fichier de sortie	Le programme sélectionne les requêtes satisfaisant ces conditions, affiche sur la sortie standard la liste des dix ressources le plus consultées et génère un fichier au format GraphViz représentant les parcours des visiteurs sur le site	Test21

4 Architecture de l'application

4.1 Classes principales

Nous avons conçu l'application pour que celle-ci s'articule autour de trois classes principales : Requete, EnsemblePages et Page. A ces trois classes principales nous avons également ajouté deux classes supplémentaires, chargées de l'interface avec le système de fichiers, RequeteDAO et EnsemblePagesDAO. Un diagramme de classe UML représentant l'agencement des classes dans l'application est disponible sur la figure 1.

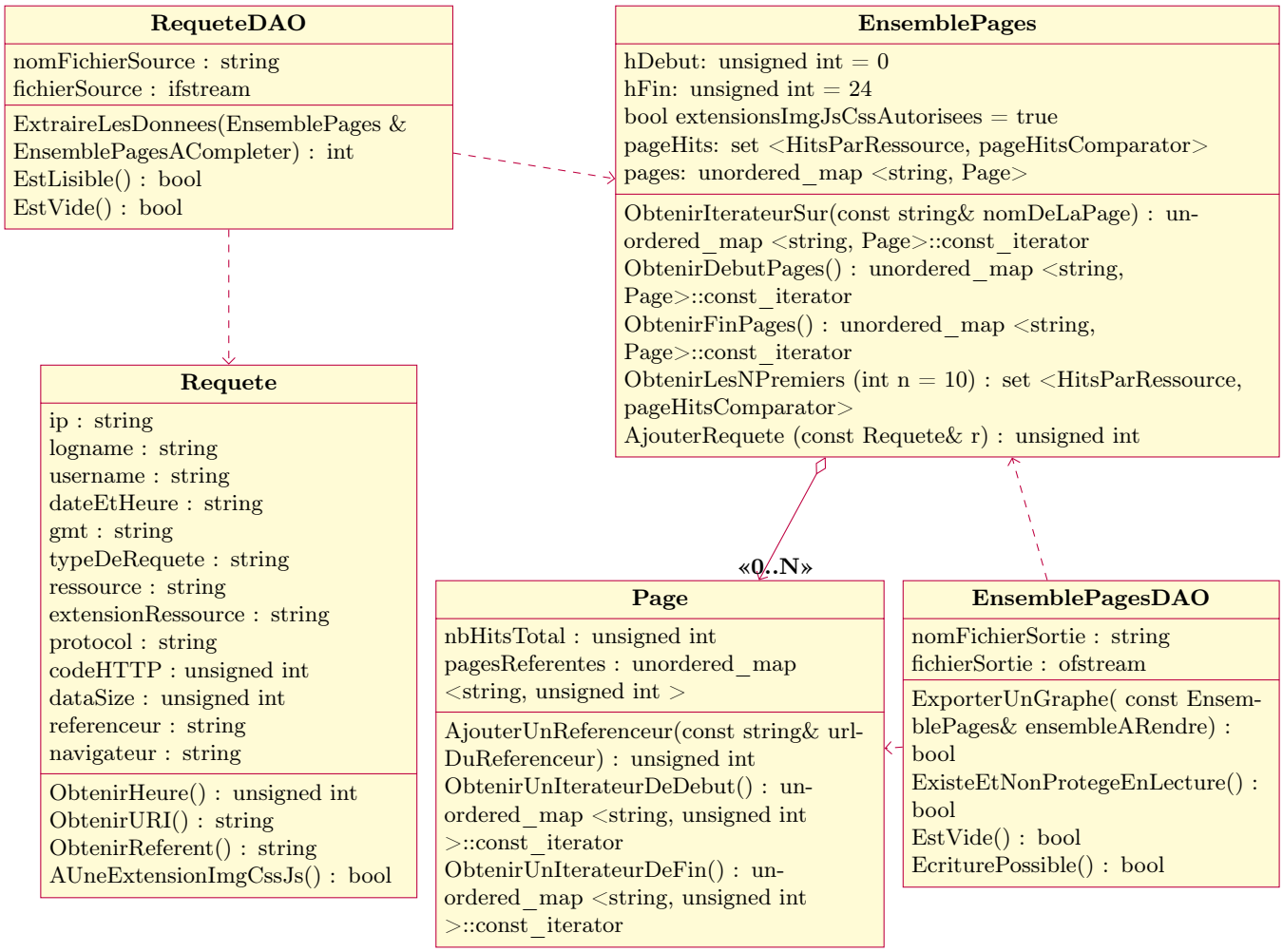

Figure 1: Diagramme de classe UML de l'application

4.1.1 Requete

La classe `Requete` se comporte comme un parseur pour une ligne de log Apache. Son constructeur par défaut `Requete (string ligneDeLog)` découpe la ligneDeLog et initialise ainsi ses attributs tels que logname, username, typeDeRequete, etc. Dans le contexte de notre application, nous avons besoin de quatre informations: la ressource, l'extension de la ressource, le référenceur et l'heure de la requête que nous faisons circuler par l'intermédiaire des méthodes `ObtenirHeure()`,`ObtenirURI()`,`ObtenirReferent()` et `AUneExtensionImgCssJs()`. C'est la classe `RequeteDAO` qui, en lisant un fichier, instancie un objet de type `Requete` à partir de la ligne du fichier que elle vient de lire. Ensuite, elle insère cet objet Requête nouvellement créé dans un objet `EnsemblePages`. Cette tâche d'extraction/insertion est faite en utilisant la méthode `int ExtraireLesDonnees (EnsemblePages & EnsemblePagesACompleter)` de `RequeteDAO`.
Le plus gros point fort de la classe Requete est sa reutilisabilité. En effet, on peut ajouter des fonctionnalités à notre application sans pour autant changer la classe `Requete` et nous pouvons

également nous resservir de cette classe au sein d'autres applications utilisant le même formatage de log.

4.1.2 Page

La classe `Page` est utilisée pour stocker et mettre à jour la structure de données associée à la liste des referenceurs d'une ressource et aussi son numero de hits total. Son constructeur `Page ( void )` instancie un objet avec un numero de hits total egal a 0. C'est ensuite à l'intermediare de la fonction `unsigned int AjouterUnReferenceur(const std::string& urlDuReferenceur)` qu'on ajoute un nouveau referenceur à cette page ou, s'il existe déjà, on incremente le numero de hits qui lui est associé.

4.1.3 EnsemblePages

La classe `EnsemblePages` est la classe principale qui assure le stockage et la mise à jour de la structure de données qui associe chaque nom de ressource avec son objet `Page` associé et de la structure de données qui classe les ressources par nombre total de hits et par leur nom. Par l'intermediare de la fonction `int AjouterRequete (const Requete& r)` qui est appelé dans `RequeteDAO`, les differentes informations sont insérées ou mises à jour dans les structures correspondantes. Au fur et à mesure de l'insertion des données en provenance du fichier de log, les ressources sont triées en fonction de leur nombre total de hits ainsi que de leur nom. A la fin de l'insertion, un appel à la méthode `TSetPagesHits ObtenirLesNPremiers(int n = 10)` permet d'obtenir le top10 des ressources les plus représentées en temps que cible des requêtes et un appel a la méthode `bool ExporterUnGraphe( const EnsemblePages & ensembleARendre)` de la classe `EnsemblePagesDAO` permet lui de générer le graphe associé à ce top10 dans un fichier `.dot`.

4.2 Structures de données utilisées

Dans les classes Page et EnsemblePages, nous avons utilisé des structures de données importées de la STL, `std::unordered_map` et `std::set`. Nous allons ici expliquer pourquoi nous les avons choisis ainsi que la maniére dont ces structures de données sont implémentées.

4.2.1 std::set

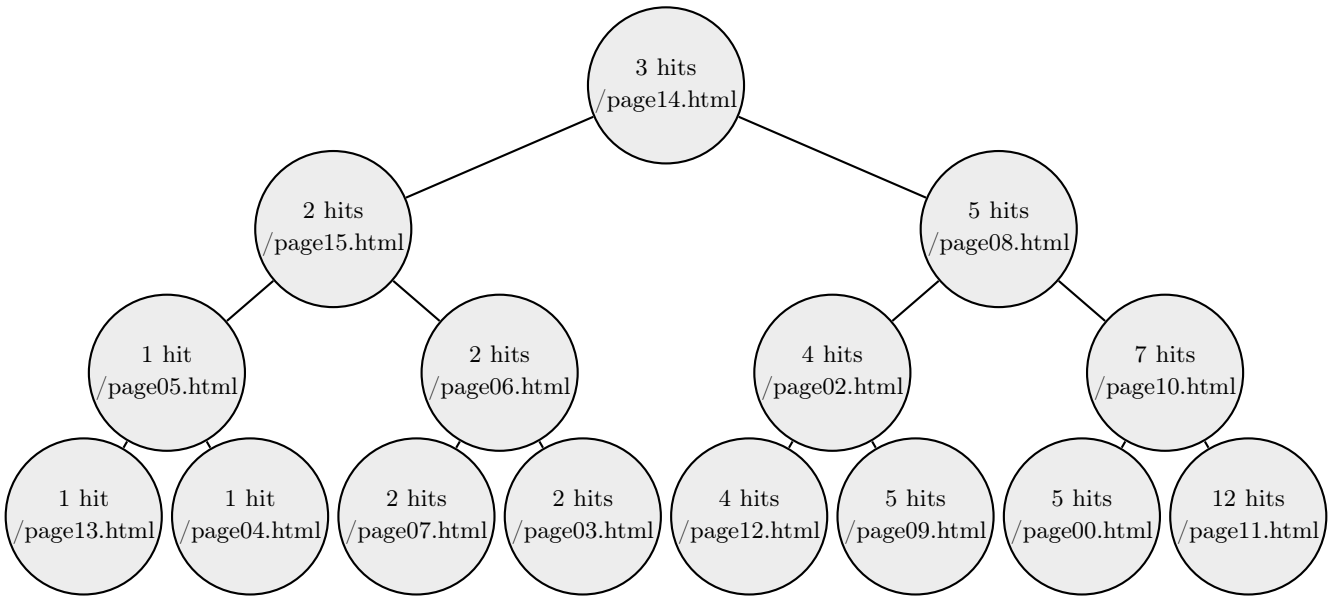

Figure 2: Schéma de l'implémentation du set pour le fichier de log de test du Test 22. On peut voir que les combinaisons d'URI et de "hits" associés sont triés par ordre de hit décroissant puis par ordre lexicographique

La classe générique `std::set` est utilisée dans la classe EnsemblePages. Elle permet de modéliser un Ensemble ordonné de valeur sans répétitions. Elle est généralement implémentée sous la forme d'un Arbre binaire, comme illustré sur la figure 2. Dans cette même classe, nous avons défini une structure nommée *HitsParRessource*. Cette structure nous permet d'associer un nombre de "Hits" à une ressource représentée par une chaîne de caractères (`std::string`). De plus, nous y avons également défini un "functor" nommé *pageHitsComparator* et permettant de comparer deux structures de type *HitsParRessource*. Ainsi, en définissant un set dans la classe EnsemblePages contenant des *HitsParRessource* et utilisant *pageHitsComparator*, cela nous permet de classer automatiquement les ressources du site internet par nombre de "Hits" décroissant puis par ordre alphabétique croissant lors de l'insertion.

Ainsi, si l'insertion est aisée, la modification des données, qui est requise si on détecte un nouveau "Hit" pour une ressource, nécessite une suppression de l'ancienne valeur puis une réinsertion avec le nouveau compte de Hits pour la ressource.

Nous avons choisi cette structure de données pour cette classe car elle offrait les avantages suivants:

- Les données insérées sont directement triées selon notre convenance selon le nombre de "Hits" de la ressource ainsi que selon son URI.
- Il y a plusieurs méthodes d'insertion et de suppression possibles avec des complexités variées. Cependant, la manière dont nous utilisons les méthodes d'insertion et de suppression nous permet d'obtenir en complexité d'insertion $\Theta(\log(\text{taille}))$, *taille* désignant le nombre d'éléments du set, dans le pire des cas et $\Theta(1)$ dans le meilleur cas. La complexité de suppression est de $\Theta(\log(\text{taille}))$ car c'est la complexité de la méthode find que nous utilisons pour la suppression (qui, ici, se fait en elle même en $\Theta(1)$).

4.2.2 std::unordered_map

La classe générique std::unordored_map est utilisée dans les classes EnsemblePages et Page. Elle permet d'associer une clef à une valeur, et ne trie pas les valeurs de manière ordonnée. Elle est généralement implémentée sous la forme d'une table de hashage, dont le schéma est visible à la figure 3

- Dans la classe EnsemblePages, nous avons défini une unordered_map avec comme clef un string et comme valeur un objet de type Page. La clef représente l'URI de la page et la Page est l'objet contenant le nombre de Hits de la page ainsi que l'ensemble des URI/URL référençant cette page. Nous avons choisi d'utiliser cette structure de données ici car elle nous permet de facilement relier l'URI d'une page du site aux informations relatives à cette page.
- Dans la classe Page, nous avons également défini une unordered_map avec comme clef un string et comme valeur associée un entier non signé. La clef représente l'URI/URL de la ressource internet faisant référence à l'objet Page en question tandis que la valeur associée est le nombre de fois que la ressource a été référencée par ce référenceur. Nous avons choisi d'utiliser cette structure de données iici car elle nous permet ici de rattacher facilement un référenceur au nombre de fois qu'il a référencé cette ressource. De plus, la structure de données nous permet de mettre très facilement le nombre de nombre de fois qu'un référenceur a référencé la ressource, et ce, sans nécessité de supprimer et de réinsérer des données dans la unordered_map. En effet, il suffit juste de trouver la clé (le nom de la ressource référente) pour pouvoir modifier le nombre associé.

Dans les deux cas, l'insertion et la recherche sont assez rapides, puisque en $\Theta(1)$ dans la majorité des cas et en $\Theta(\text{taille})$ dans le pire des cas. La suppression est également dans le pire des cas en $\Theta(\text{taille})$, mais ce n'est pas un problème car nous n'avons pas besoin de supprimer d'éléments dans les deux unordered_map que nous utilisons.

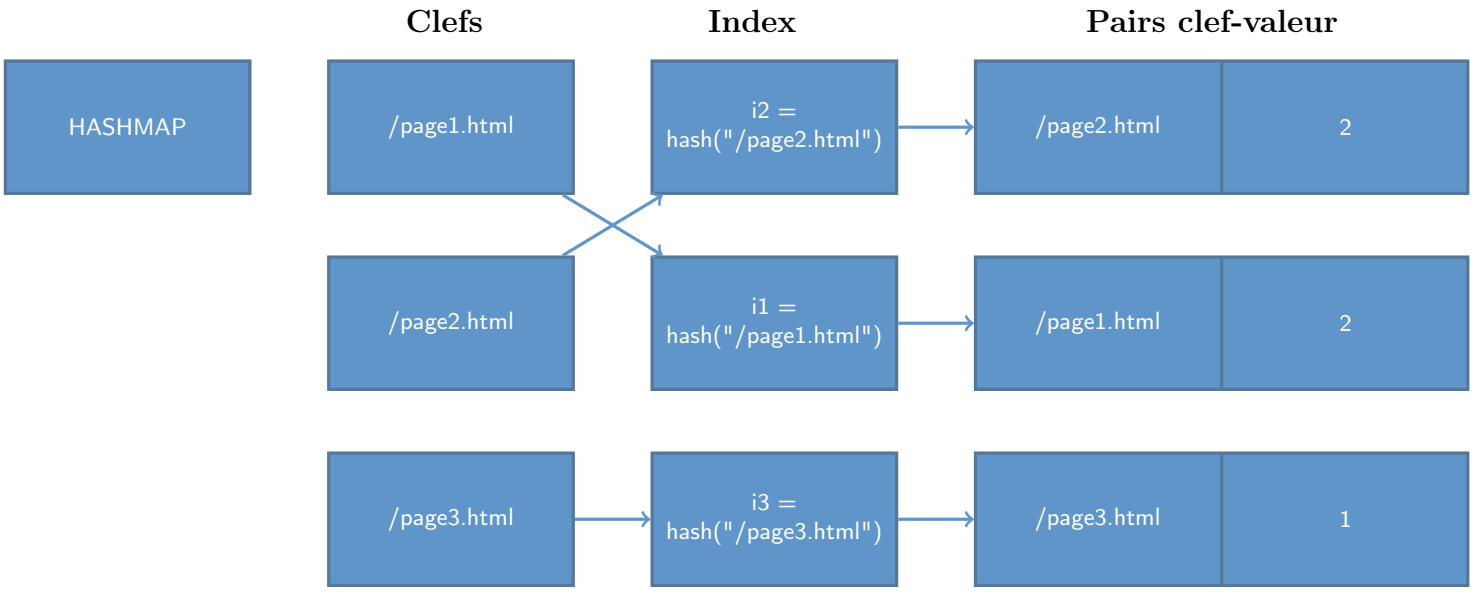

Figure 3: Schéma de l'implémentation du unordered_map de l'objet `Page` associé à la ressource `"/page1.html"` pour le fichier de log de test du Test 17.

5 Conclusion

A travers ce TP, nous avons pu utiliser de manière intensive les ressources de la STL telles que ses structures de données ainsi la classe string. Les structures de données génériques de la STL, une fois choisies et personnalisées pour répondre à nos besoins, nous ont permis de répondre efficacement à nos besoins tout en nous permettant de bénéficier d'optimisations algorithmiques qu'il n'a pas été nécessaire pour nous de développer, comme le fait de structurer les données dans une table de hachage ou dans un binaire de recherche. Une amélioration de l'application reste néanmoins possible. Nous pourrions, par exemple, exploiter les possibilités avancées de GraphViz en terme de rendu de Graphe.