Auteurs : DUPONT Yann - MIGNOT Thomas Groupe : 3IF4

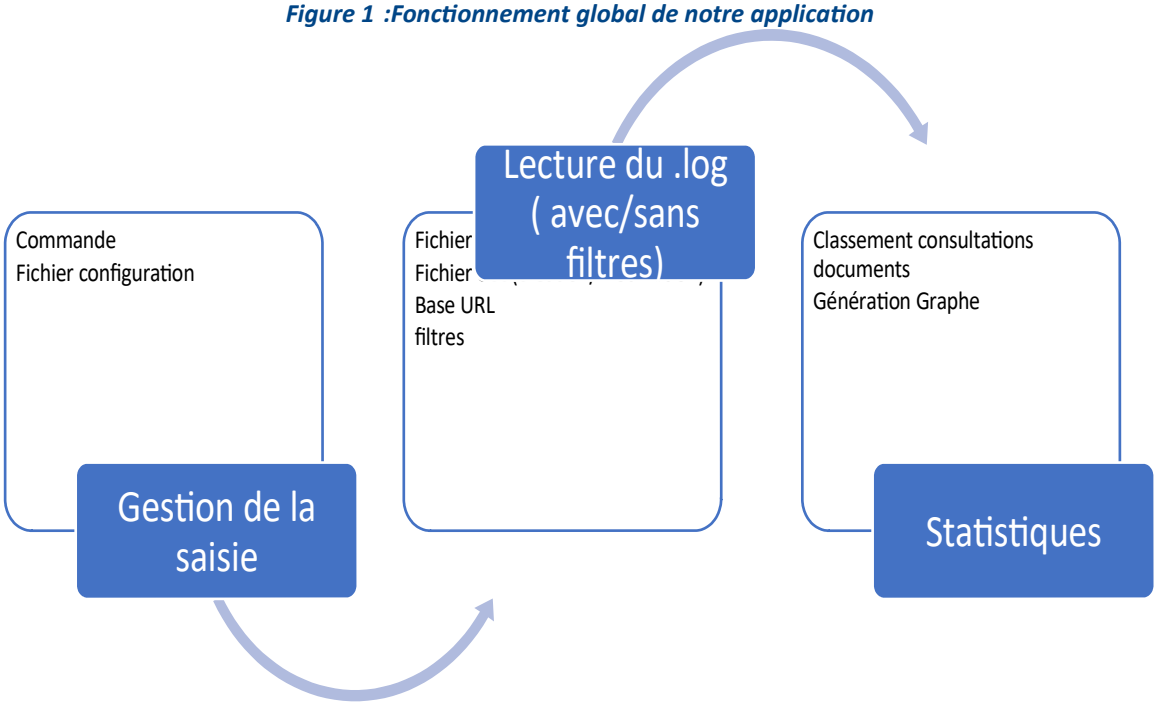
Compte rendu TPC++ n°4 : Analyse de logs Apache

I. Spécifications de notre application

I.1. Description générale

Notre application permet de lire des fichiers de logs Apache et de faire par défaut le classement des documents les plus consultés (10 premiers par défaut). D'autres options sont possibles comme produire un fichier .dot permettant par la suite de construire un graphe avec pour nœuds les documents et pour arcs leurs transitions. Il est également possible d'appliquer des filtres : un sur l'heure de la requête et un qui permet d'exclure les documents au format image, css ou javascript. Aussi, un fichier de configuration en format texte peut être ajouté afin d'éliminer l'entête de l'URL (du référant) lorsqu'elle est locale. Cet entête doit être recopié dans le fichier texte correspondant pour être prise en compte.

Figure 1 :Fonctionnement global de notre application



I.2. Spécifications détaillées

Les options peuvent être spécifiées dans n'importe quel ordre. Nous les avons définis conformément à l'énoncé du TP :

[-g nomfichier.dot] afin de produire un fichier au format GraphViz

[-e] afin d'exclure tous les documents qui sont définis comme une image, css ou javascript c'est-à-dire les formats suivants : css, js, png, jpg, gif, bmp.

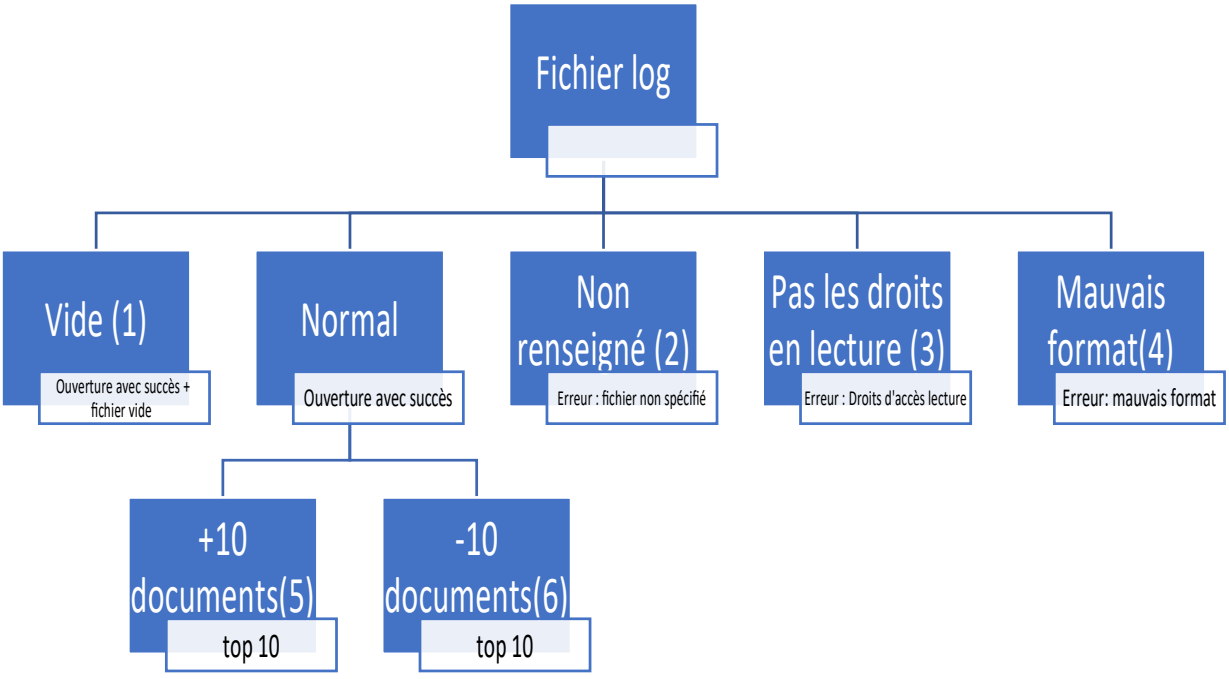
[-t heure] qui permet de ne sélectionner que des hits dans le créneau horaire correspondant à l'intervalle [heure, heure+1[. L'heure saisie doit appartenir à l'intervalle [0,23].

Une fois les options saisies, la commande devra se terminer par le nom du fichier log à lire. Ce dernier doit exister et l'utilisateur doit disposer des droits en lecture pour que le programme continue.

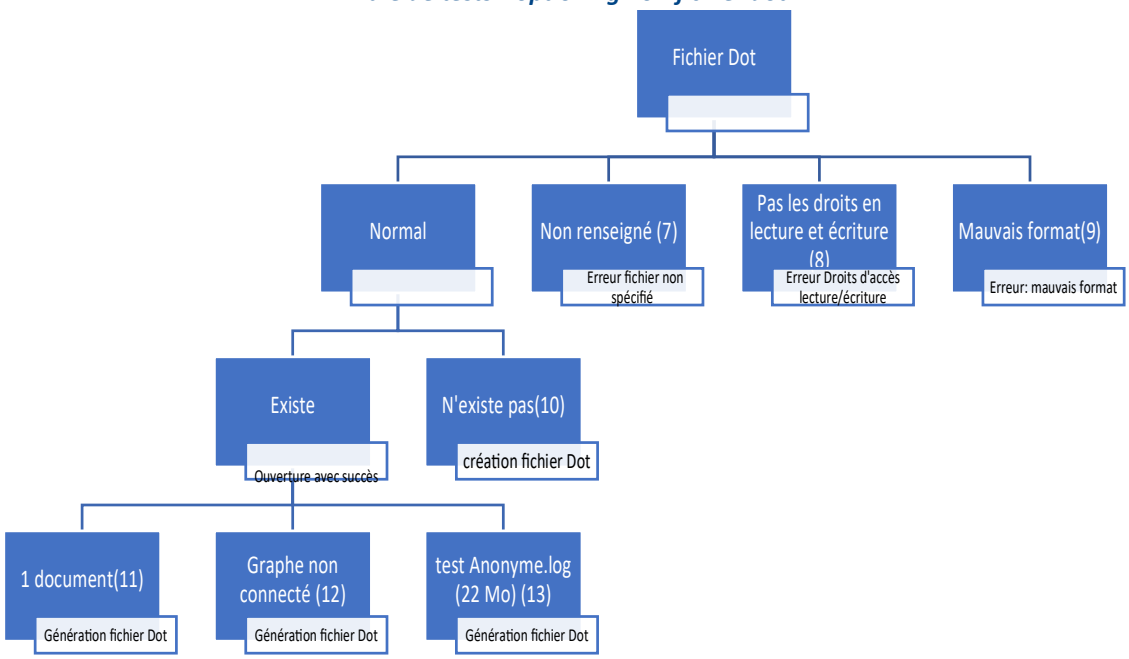
Format de la commande
./analog [options] nomfichier.log

Nous avons géré tous les cas limites que nous pouvons regrouper les arbres ci-dessous. Les feuilles représentent chacun un test avec un numéro précisé qui sera présent dans le dossier Tests.

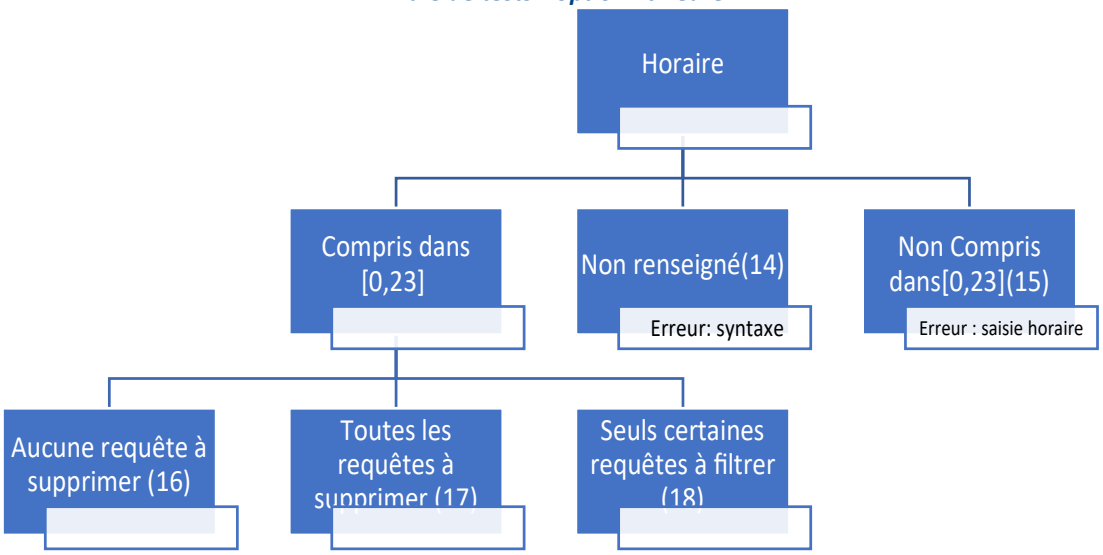
Arbre de tests : aucune option



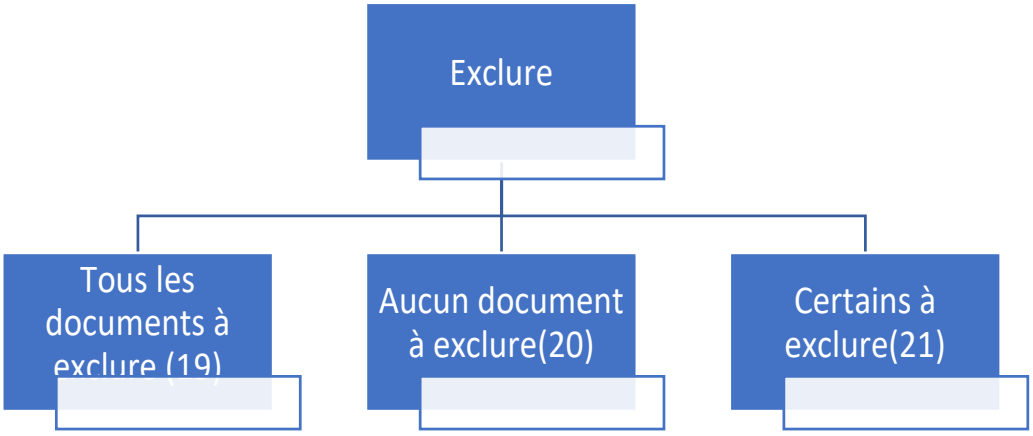
Arbre de tests : option –g nomfichier.dot



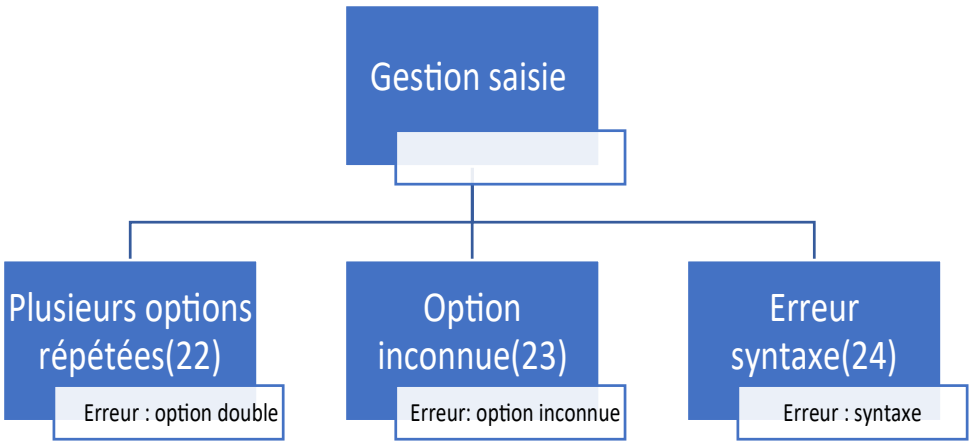
Arbre de tests : option –t heure



Arbre de tests : option –e



Arbre de tests : erreurs saisies



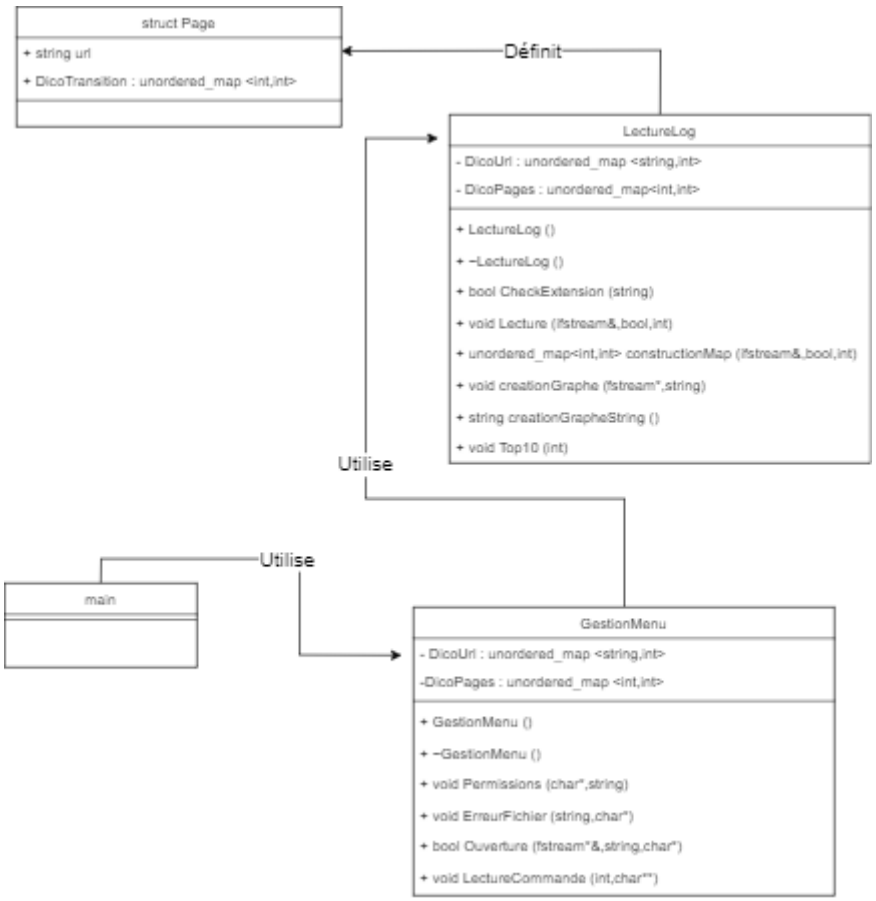
II. Architecture Globale de l'appplication

Nous avons structuré notre application en 3 classes différentes :

- Main qui permet de récupérer les arguments
- GestionMenu qui a pour rôle de gérer toutes les erreurs de saisies et tous les cas différents d'options.
- LectureLog qui permet de lire les fichiers logs, d'avoir le classement des documents les plus consultés et de générer le graphe. Cette classe contient une structure Page contenant des informations comme l'URL, le nombre de clics et une collection pour gérer les transitions arrivant sur la page.

Il nous a paru judicieux de placer la lecture et les statistiques au sein de la même classe car cela la rend totalement indépendante. Il est ainsi possible de l'utiliser uniquement pour lire le fichier car le découpage en méthode le permet et rend chaque utilisation possible sans nécessiter d'autres classes. Nous aurions pu la découper en deux classes différentes (en évitant l'interdépendance) avec une permettant uniquement la lecture et une pour les statistiques mais nous avons jugé que ce n'était pas nécessaire car la classe Lecture n'aurait été constituée que d'une vrai fonction membre.

Nous n'avons pas ajouté une classe permettant de sauvegarder toutes les informations relatives aux requêtes (date, heure, userID etc…) car ce n'est pas l'objectif du TP mais il serait facile de rajouter la sauvegarde au moment de la lecture du fichier log.



Rajouter Schéma UML

III. Structures de données

Nous avons d'abord eu pour idée d'éviter de manipuler des éléments couteux sur les clefs de nos structures associatives comme par exemple une structure Page. Ainsi, nous avons mis en place une première collection de type tableau associatif afin d'identifier l'url d'un document uniquement par un indice unique. Ce numéro d'indice n'est pas un attribut, il est fictif mais il nous sert à éviter de faire des recherches sur des clés plus contraignantes qui vont nécessiter des fonctions de hachage plus coûteuses.

Ainsi, par la suite, nous n'aurons plus que des entiers dans les clefs de nos structures des données. C'est aussi cette collection qui testera la présence ou non d'une url afin de savoir s'il faudra l'ajouter ou non.

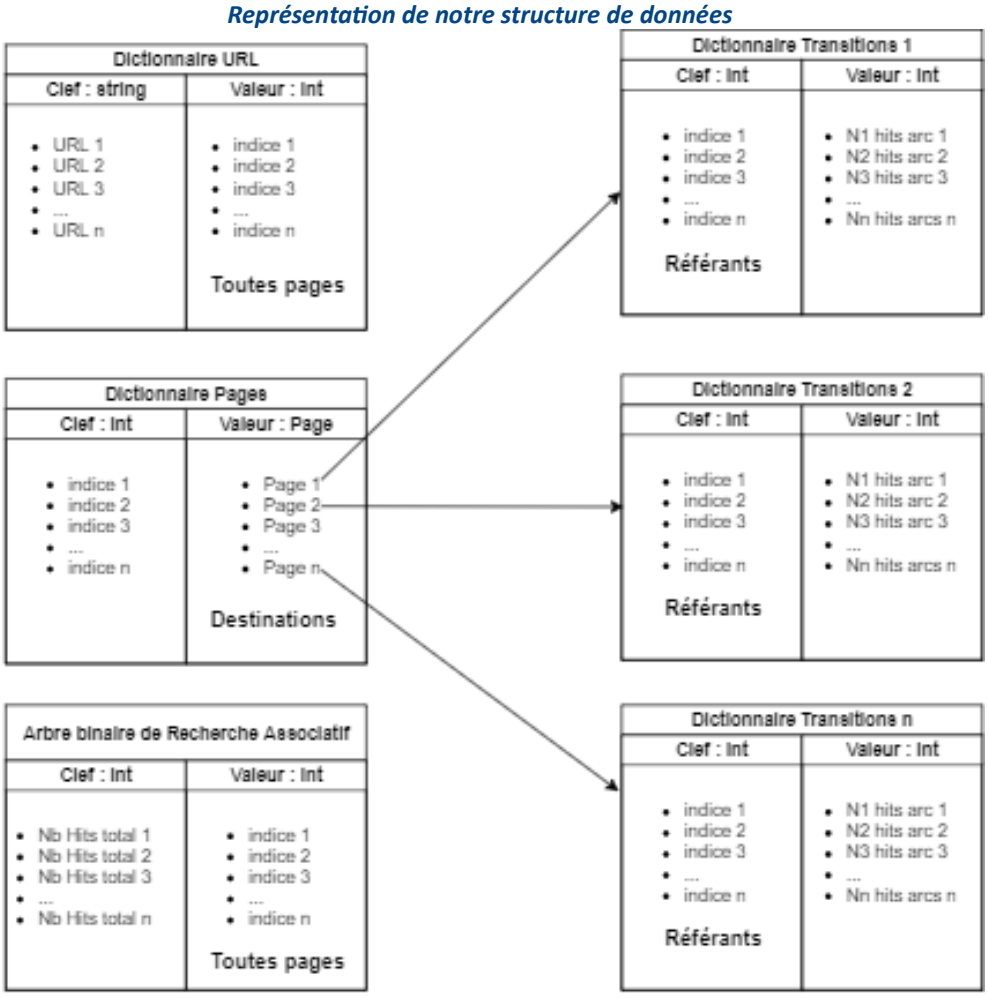
Pour gérer l'ensemble des documents, il faut que l'insertion soit peu coûteuse car on risque de réaliser cette opération beaucoup de fois. Ainsi, nous avons opté également pour une structure de type dictionnaire non ordonné.

Il nous manque encore un moyen de gérer les transitions. Maintenant que nous avons défini des indices fictifs sur les pages, il paraît évident de les réutiliser en optant pour un système clé/valeur avec pour clé l'indice de la page source et pour valeur le nombre de hits correspondant à cette transition. L'accès à la valeur correspondante à n'importe quelle clef est encore prioritaire ce qui nous fait de nouveau opter pour un tableau associatif.

Nous avons donc 3 collections qui permettront de gérer l'ensemble des éléments nécessaires au graphe. Afin de faire le classement des pages les plus consultées, nous devons cette fois changer de structure car nous allons devoir faire de la recherche en fonction du nombre de hits. Nous avons opté pour un arbre binaire de recherche associatif avec un système clef/valeurs. Ainsi l'arbre serait trié par valeur de clef. Comme nous voulons trier les documents en fonction du nombre de hits, la clef sera définie par le nombre de hits et la valeur par l'index du document correspondant. Il est à noter qu'il peut y avoir le même nombre de hits donc les mêmes clefs) avec des valeurs différentes d'index. C'est pour cette raison que nous n'utiliserons pas une map mais une multimap dans la STL.

Lorsque nous ferons le classement des pages les plus consultées, comme l'arbre sera trié, la recherche des maximums sera en O(1). L'insertion quant à elle sera en log(n).

Représentation de notre structure de données



Maintenant que nous avons les structures théoriques, nous pouvons les appliquer au cas de la STL :

-Dictionnaire URL : unordered_map <string,int>

-Dictionnaire Pages : unordered_map <int,Page>

-Dictionnaire Transitions : unordered_map <int,int>

-Arbre recherche associatif : multimap_map <int,int>

IV. Conclusion

Ce TP nous a permis de nous familiariser avec les conteneurs et algorithmes proposés dans la STL. Nous avons aussi conçu une application plus robuste que dans les anciens TP avec une gestion de la saisie mieux gérée et la mise en place de tests automatisés.