Université Paris Diderot
L3 Informatique

Conduite de Projet
Année 2016-2017

Ungoliant : Robot d'Indexation et Moteur de Recherche

version 1.1

1 Introduction

> *Un* moteur de recherche *est une application web permettant, de trouver des ressources à partir d'une requête sous forme de mots. [...] Ce sont des instruments de recherche sur le web sans intervention humaine, ce qui les distingue des annuaires. Ils sont basés sur des « robots », encore appelés bots, spiders, crawlers ou agents qui parcourent les sites à intervalles réguliers et de façon automatique pour découvrir de nouvelles adresses (URL). Ils suivent les liens hypertextes qui relient les pages les unes aux autres, les uns après les autres. Chaque page identifiée est alors indexée dans une base de données, accessible ensuite par les internautes à partir de mots-clés.*
>
> — https://fr.wikipedia.org/wiki/Moteur_de_recherche

Le but de ce projet est d'implémenter en langage C un *moteur de recherche*, un programme capable d'explorer récursivement le contenu d'un ensemble de pages HTML ou de sites Web, de les indexer, et de permettre à un utilisateur d'effectuer une recherche sur leur contenu.

Le projet comprend quatre parties principales. Il vous faudra implémenter :

1. un *analyseur syntaxique* (parser) de pages HTML ;
2. un gestionnaire de *fichier d'index*, ce dernier contenant les informations nécessaires pour associer un ensemble de pages à un mot-clef donné, le gestionnaire permettant d'effectuer, ajouter, supprimer ou mettre à jour de telles associations ;
3. un *robot d'indexation* (crawler), ou *araignée*, qui télécharge chaque page, analyse leur contenu à l'aide du parser, ajoute de l'information à l'index et se déplaçe de page en page en suivant les liens externes ;
4. enfin, un *gestionnaire de requêtes* (query engine) permettant à un utilisateur de retrouver des pages indexées à partir d'un ou plusieurs mots-clefs.

Le projet est à faire en trinômes. La progression régulière de votre travail au cours du semestre, de même que le bon usage des outils et des pratiques de conduite de projet présentés dans ce cours, seront au moins aussi importantes pour votre évaluation que le résultat final. Les informations pratiques utiles vous seront fournies sur la page Moodle du cours, http://moodlesupd.script.univ-paris-diderot.fr/course/view.php?id=6660

2 Réalisation du projet

2.1 L'analyseur syntaxique (parser)

L'analyseur syntaxique est destiné à extraire le contenu pertinent d'une page HTML : dans notre cas, l'ensemble des liens externes et le texte brut (séparé en mots), c'est-à-dire tout ce qui

ne relève pas du balisage HTML.

Votre analyseur devra faire preuve d'une certaine tolérance – il est assez fréquent, en pratique, qu'une page HTML soit (très) mal formée.

2.2 Le robot d'indexation (crawler)

Le robot commencera son exploration à partir d'une ou plusieurs URLs – il devra gérer au moins les URLs de la forme `http://`, `https://` et `file://`. Il commencera par télécharger les données d'une URL, par exemple à l'aide de la librairie `libcurl` [8] ou d'un autre outil de votre choix (regardez par exemple [1]), puis les transmettra au parser.

Le robot mettra ensuite à jour l'index (voir ci-dessous) en fonction de tous les mots rencontrés dans le texte brut de la page. Il continuera récursivement son exploration en suivant tous les liens externes des pages, jusqu'à une certaine limite qui lui sera donnée (profondeur maximale, nombre total de pages visitées, ...). Il construira également le graphe du réseau visité : les noeuds correspondent aux URL visitées et les arêtes (dirigées) aux liens externes. Ce graphe sera conservé dans un fichier externe, dans un format que vous choisirez.

A noter que le robot devra pouvoir suivre non seulement des URLs complètes, mais également des URLs locales, *eg.* de la forme «`../page.html`».

2.3 L'index

L'*index* est un fichier externe qui associe chaque mot rencontré à la liste de pages dans lesquelles il apparaît. Il est produit par le robot et utilisé par le gestionnaire de requêtes.

Le format de l'index est libre. Vous pouvez faire le choix de vous servir d'une base de données standard, *eg.* gérée à l'aide de la librairie `sqlite` [15], mais nous vous suggérons plutôt de trouver votre propre format d'index.

Ce format n'a pas besoin d'être complexe : le fichier pourra ne contenir qu'une simple suite de lignes formées d'un mot suivi d'une liste d'URLs, éventuellement complétées du nombre d'occurrences de ce mot dans chaque page – ou de toute autre information utile aux recherches futures (classement, pertinence de l'association, mots proches, etc).

Le contenu de ce fichier pourra dans ce cas être chargé en mémoire dès le lancement du programme, et stocké dans des structures adaptées à la recherche rapide d'associations (arbres binaires de recherches, arbre lexical, table de hachage [5, 6]...). Les données seront mises à jour par des primitives travaillant directement sur ces structures, puis sauvegardées sous leur format original.

2.4 Le gestionnaire de requêtes (query engine)

Le gestionnaire de requêtes devra permettre au moins de retrouver dans l'index l'ensemble des pages contenant un mot donné (*cf.* la section 3 pour des suggestions d'extensions) avec une certaine marge de tolérance – insensibilité aux majuscules/minuscules, à la présence ou absence d'accents, etc. L'ordre de présentation des résultats se ferant suivant leur pertinence, *eg.* en commençant par les pages contenant le plus d'occurrences du mot cherché.

La forme de l'interface utilisateur est libre : ligne de commande, interface graphique [14, 13], interface web ...

3 Extensions possibles

Tolérance aux pages malformées La plupart des moteurs de recherche sont capables d'indexer des pages Web malformées (p.ex. tags non fermés). Pour maximiser la robustesse de votre moteur de recherche vous pouvez utiliser plusieurs techniques et librairies externes, comme p.ex. `libxml` [11] (via son parser html), `beautifulsoup` [2] (en Python), ou même simplement `libpcre` [9] pour un « parsing» basé sur des expressions régulières.

Optimisation de la mise à jour de l'index Quand le robot explore une page Web qu'il a déjà visitée lors d'une exploration antérieure, on voudrait qu'il ne mette à jour l'index et le graphe du réseau que si la page a été modifiée depuis la dernière visite - par exemple en utilisant sa *somme de contrôle* (checksum) [16].

Si la page n'a pas été modifiée, on ne la parsera pas, et on utilisera le graphe du réseau pour explorer récursivement ses liens externes.

Recherche de fichiers Le robot pourra étendre son champ de recherche aux noms des fichiers rencontrés ; on pourra ainsi faire des recherches par type de fichier, etc. Vous pouvez également étandre la recherche aux attributs HTML (texte alternatif des images, description des pages ...) de la manière qui vous semblera pertinente.

Affichage du graphe du réseau Une fois le réseau exploré par le robot, on pourra tenter d'en offrir une représentation graphique. On voudrait pouvoir par exemple afficher le plan (*sitemap*) d'un site individuel, ou bien afficher le réseau entier, par exemple en utilisant la bibliothèque `graphviz` [10].

PageRank Les pages pourront se voir attribuer un score suivant un algorithme *à la* PageRank [12] pour influencer l'ordre des résultats.

Recherche d'entrées proches La recherche d'un mot peut prendre en compte des entrées dans l'index non strictement identiques à ce mot, mais suffisamment proches : majuscules, accents, erreurs de frappe, pluriels, conjugaisons, etc. Pour les erreurs de frappe on pourra par exemple utiliser la distance de Levenshtein [3] pour mesurer le degré de similarité entre deux mots.

Langage de recherche étendu, recherches multiples Le gestionnaire de requêtes pourra accepter plusieurs mots, en considérant comme les plus pertinentes les pages contenant tous ces mots. Il pourra aussi accepter des requêtes formées à l'aide de connecteurs booléens, *eg.* «`((foo AND bar) OR baz) AND NOT quux`» ou bien en s'inspirant des opérateurs de recherche de Google [7].

Nuage de mots Vous pouvez générer un nuage de mots qui représente visuellement la fréquence relative des mots qui apparaissent dans l'ensemble des pages d'un site. Vous choisirez sous quel forme générer le nuage, par exemple comme une page HTML. On pensera à ignorer les mots trop communs (articles, etc.).

4 Évaluation de votre projet

La notation prendra en compte tous les points suivants.

Utilisation des outils standard. Les cours magistraux présenteront tout au long du semestre des outils standard de développement (Makefile, git, etc.) qui doivent être *effectivement* utilisés pendant le projet. Cette utilisation va de pair avec une séparation des différentes parties du projet en modules et librairies séparées.

Par exemple, une utilisation frauduleuse de *Makefile* pourrait être un appel unique à un script externe.

Modularité et tests. Les différents modules du programme devront être séparés en différents fichiers-sources (`.c`) et en fichiers d'en-tête (`.h`) utilisés à bon escient. Vous préparerez aussi des tests unitaires de chaque fonction importante sur des exemples choisis, et des tests d'acceptation des fonctionnalités globales du projet. Ces tests seront inclus dans votre dépôt git.

Contrôle continu. Le travail doit être réalisé tout au long du semestre grâce à un répertoire *git* pour chaque trinôme. Dès les premières semaines, vous donnerez accès à ce répertoire à tous les enseignants. Ceci permettra notamment de garder la trace et les dates de contributions de chaque étudiant. Vous pouvez choisir entre le service d'hébergement GitLab offert par l'université http://moule.informatique.univ-paris-diderot.fr:8080/ et tout autre service tiers, toujours à la condition d'y donner accès à vos enseignants.

Journal. Chaque trinôme tiendra à jour un *journal de développement* inclus dans son répertoire git. Chaque semaine, vous indiquerez dans votre journal vos objectifs pour la semaine à venir, et vos résultats de la semaine passée. Dans la mesure du possible, faites en sorte que chaque commit corresponde à une tâche précise et soit effectué par la personne ayant effectué la tâche. Un dépôt git n'est pas limité à du code, n'hésitez pas à y inclure d'autres documents qui nous permettront de suivre votre travail, par exemple : des courts résumés de prise de décision, des notes sur le fonctionnement des bibliothèques que vous allez utiliser, la structure que vous projetez pour vos modules, etc.

Lisibilité et clarté. Il est fortement recommandé d'indenter systématiquement, d'éviter les lignes trop longues (80 caractères), de choisir des noms parlants pour les variables et fonctions, et de commenter le code souvent, mais en restant concis. Vous supprimerez également toute duplication de code, suivant le principe DRY [4] ("Don't Repeat yourself").

Références

[1] Alternatives à libcurl. https://curl.haxx.se/libcurl/competitors.html.

[2] beautifulsoup. http://www.crummy.com/software/BeautifulSoup/bs4/doc.

[3] Distance de Levenshtein. http://fr.wikipedia.org/wiki/Distance_de_Levenshtein.

[4] Don't repeat yourself. https://en.wikipedia.org/wiki/Don%27t_repeat_yourself.

[5] glib – hash tables. https://developer.gnome.org/glib/stable/glib-Hash-Tables.html.

[6] Gnu c library – hsearch. http://www.gnu.org/software/libc/manual/html_node/Hash-Search-Function.html.

[7] Google search operators. https://support.google.com/websearch/answer/2466433.

[8] libcurl. http://curl.haxx.se/libcurl.

[9] libpcre. http://www.pcre.org.

[10] Librairie graphviz. http://www.graphviz.org/pdf/libguide.pdf. On peut trouver des exemples d'utilisation ici : http://www.graphviz.org/Documentation.php.

[11] libxml2. http://xmlsoft.org/html/index.html.

[12] page rank. http://fr.wikipedia.org/wiki/PageRank.

[13] Sdl (wikipedia). http://en.wikipedia.org/wiki/Simple_DirectMedia_Layer.

[14] simple directmedia layer (sdl). http://www.libsdl.org.

[15] sqlite. http://sqlite.org.

[16] wikipedia – somme de contrôle. https://fr.wikipedia.org/wiki/Somme_de_controle.