Data mining : intro et un premier web crawler

---

1. Intro -- généralités sur le web crawling

Le web est une collection immense de pages -- noeuds contenant de l'information -- reliés entre eux avec les hyperliens. On appelle 'web crawling' le balayage (de tout ou une partie) de ces noeuds, à l'aide d'un programme qui télécharge donc ces informations noeud par noeud, et qui découvre et ensuite explore ces liens.
Le but en est donc de récupérer ces informations, pour les filtrer, classifier, indexer, mettre dans des bases de données, et donc les exploiter par la suite.
Un web crawler dispose d'une liste d'URLs qu'il doit visiter. Au démarrage, on donne au web crawler quelques URLs de départ. Cette liste est une queue, dont les URLs sont extraites un par un et visités. La liste est re-remplie ensuite avec les nouvelles URLs découvertes dans les pages téléchargées et filtrées, et le processus continue jusqu'à atteindre la profondeur maximale.

Les étapes principales du processus de web crawling sont les suivantes :

1. initialisation de la queue des URLs avec les URLs de départ
2. tant qu'on peut encore visiter
    1. prendre l'URL disponible suivant depuis la queue
    2. télécharger le contenu et marquer l'URL comme visitée
    3. extraire les hyperliens depuis le document nouvellement téléchargé et les rajouter dans la queue s'ils satisfont les critères nécessaires
    4. réévaluer les conditions pour continuer à visiter des sites (profondeur maximale, nombre maximal de documents récupérés, temps maximal d'exécution, queue d'URLs vide, etc.)
    5. attendre "un peu" avant de continuer (pour ne pas "assommer" le serveur)

Parmi les critères à satisfaire par un nouvelle URL pour qu'elle soit rajoutée dans la queue, on peut mentionner la vérification du fichier `robots.txt` qui indique les chemins qu'on souhaite que les crawlers n'explorent pas, par exemple ainsi :

```
User-agent: *
Disallow: /data/blah.html
```

Ce fichier est d'habitude placé à la racine du site, donc ainsi :

```
http://www.truc.com/robots.txt
```

2. Construire un petit web crawler

Nous allons utiliser du code Java existant et écrire un web crawler assez simpliste. Nous allons ensuite le tester en "local", en le faisant parler a un mini server http (toujours en Java), qui vous est fourni et qui est à faire tourner par vous-même.

2.1 "Vue d'en haut"

Le code Java que nous manipulerons ici vit dans quelques fichiers qui se trouvent dans deux répertoires principaux : `server` et `crawler`. Tous ces fichiers Java étant de petite taille et peu nombreux, on peut bien travailler avec n'importe quel éditeur, même `vi`, et donc sans nécessairement passer par `eclipse`.
Pour compiler, nous disposons également d'un `makefile` "traditionnel", qui appelle `javac` et puis `jar`.

Travail à faire

- Créez un répertoire général pour ce cours, appelé `datamining`, et à l'intérieur, créez un répertoire pour ce tp, appelé `tp1`.
- Téléchargez dans `tp1` l'archive avec le code initial et installez-la (en faisant par exemple `zcat archTp1Code.tgz | tar xvf -`). Vous allez y trouver un nombre de répertoires, dont le plus important pour la suite du travail est `crawler`.
- Faites un `ls -l` et assurez-vous que vous voyez un `makefile`, un répertoire nommé `server`, un autre nommé `crawler`, etc.
- Lancez `make` (sans aucun autre argument). Suite à la compilation et à la création de l'archive `server.jar`, le serveur sera disponible pour lancement (et le squelette du crawler sera également compilé)
- Testez le serveur ainsi
    1. Lancez le serveur avec (depuis le même shell où vous avez fait `make`) la commande

       ```
       java -jar server.jar 8123
       ```

    2. Ouvrez un nouvel onglet ou une nouvelle fenêtre de votre browser préféré
    3. Allez avec le browser à l'adresse suivante

       ```
       http://localhost:8123/data/indexClean.html
       ```

    4. Naviguez à partir de cette page très simple, et vérifiez que dans le shell ou le serveur avait été lancé, le serveur affiche au fur et à mesure tout ce qu'il renvoie au client. Pour l'arrêter on doit faire Ctrl-C.
- Dans un autre shell, toujours depuis le répertoire `tp1`, allez dans le répertoire `crawler`. Faites un `ls -l` et écrivez sur une feuille de papier (à rendre à la fin à votre enseignant)
    1. la liste des fichiers (et donc la liste des classes) du crawler
    2. la liste des méthodes (constructeur compris) de `SimpleCrawler` appelées dans la fonction `main()` (regardez bien entendu dans les fichiers `.java`).
    3. les pas essentiels de la fonction `main()`
    4. la donnée membre de `SimpleCrawler` qui contient le nombre maximal d'URLs à visiter
    5. le rôle de la méthode `crawl()` en regardant son code dans le fichier ainsi que la section 1. Intro du texte de ce TP.
    6. la donnée membre de `SimpleCrawler` qui contient la queue des URLs à visiter

7. la donnée membre de `SimpleCrawler` qui contient la liste des URLs déjà visités (étape 2 de la boucle 2 de la section 1. Intro).
8. la méthode qui rajoute un URL à la liste des URLs déjà visités, et le marque également comme visité
9. le rôle de la méthode `CrawlerUrl.setRawContent()` (la documentation de `Jsoup` se trouve à `http://jsoup.org/`).
10. les noms des trois méthodes de la classe `CrawlerUrl` qui rendent les résultats de l'analyse avec `jsoup`, et ce qu'elles rendent effectivement

2.2 Code à écrire dans `SimpleCrawler.java`

Travail à faire (sauvegardez et compilez souvent, et rajoutez des 'print')

1. Écrivez la méthode `SimpleCrawler.continueCrawling()` : elle doit répondre affirmativement si et seulement si la queue des URLs à visiter n'est pas vide (méthode `Queue.isEmpty()`), et si on n'a pas encore visité plus d'URLs qu'on a dit au constructeur de `SimpleCrawler`.
2. Finissez le corps de la méthode `SimpleCrawler.getContent(CrawlerUrl url)`

```
     private String getContent(CrawlerUrl url) { // methode
essentielle --
        // recuperation du fichier .html depuis le serveur
  HttpClient httpclient = new DefaultHttpClient();
  String text = new String();
        try {
            HttpGet httpget = new HttpGet(url.getUrlString());
//construction
      //  de l'objet qui fera la connexion

            System.out.println("executing request " +
httpget.getURI());
            // construction de l'objet qui gerera le dialogue avec le
serveur
            ResponseHandler responseHandler =
          new BasicResponseHandler();
            text = httpclient.execute(httpget, responseHandler); //et
on y va
            System.out.println("----------------------------------
---");
            System.out.println(text);
            System.out.println("----------------------------------
---");
  }
  catch(Throwable t) {
      System.out.println("OOPS YIKES "+ t . toString());
      t . printStackTrace();
  }
  finally {
            // Lorsque on n'a plus besoin de l'objet de type
HttpClient
            // on ferme la connexion pour eliberer rapidement les
resources
            // systeme qu'on avait monopolisees
            httpclient.getConnectionManager().shutdown();
        }
  // appeler la methode de SimpleCrawler qui marque l'URL comme
visite
```

```
  // et qui l'insere dans la liste des URLs visites

  // appeler la methode de CrawlerUrl qui recoit le texte HTML brut
  //  (et le donne au parseur jsoup, pour en extraire le texte, le
titre,
  //   les liens, etc,); l'objet CrawlerUrl a utiliser est 'url'
        return text;

     }
```

3. Écrivez sur la feuille de papier l'utilité du `try-catch` dans `getContent()` par rapport à ce qui peut se passer concernant les connexions, les liens, etc.
4. Dans le shell, toujours depuis le répertoire `tp1`, allez dans le répertoire `petitExemple`. Faites `ls -l *_1.txt` et écrivez sur la feuille de papier les noms de fichiers que vous voyez apparaître. Regardez le contenu du fichier `download_1.txt`. Assurez-vous que votre serveur tourne toujours dans l'autre shell, et allez avec le browser à l'adresse

   `http://localhost:8123/data/indexClean.html`

   et cliquez sur `biz-01`. Comparez le texte qu'affiche le browser et le contenu du fichier `download_1.txt` : la seule différence est le formatage.

5. Regardez maintenant le contenu du fichier `title_1.txt` -- c'est bien le titre du document.
6. Regardez maintenant le contenu du fichier `url_1.txt` -- c'est bien l'url du document.
7. Faites maintenant `ls -l url_*.txt` et regardez le contenu de ces fichiers, pour les numéros 1, 2, et 3. Ces numéros sont ce qu'on appelle ***IDENTIFICATEUR DE DOCUMENT*** (document id). Dans le code, la donnée membre `numberItemsSaved` est celle qui numérote les documents au fur et à mesure. Écrivez sur la feuille de papier le nom de la méthode de `SimpleCrawler` qui l'incrémente.
8. Finissez le corps de la méthode `SimpleCrawler.saveContent(CrawlerUrl url)`, sachant que `outputPath` est une donnée membre qui est déjà remplie par le constructeur.

```
   private void saveContent(CrawlerUrl url)  throws Exception {
String fileId = String.valueOf(numberItemsSaved);

BufferedWriter rankOutput     =
    new BufferedWriter(
        new FileWriter(outputPath + "/rankscore_"+ fileId +".txt"));
rankOutput     . write("0.0");
      rankOutput     . flush();
      rankOutput     . close();

String docContent;
String docTitle;

// TRAVAIL A FAIRE

// recuperez dans docContent le texte extrait avec jsoup de la
// variable url, en utilisant une methode de CrawlerUrl

// recuperez dans docTitle le titre extrait avec jsoup de la
// variable url, en utilisant une methode de CrawlerUrl
```

```
        this.crawlOutput.append( url.getUrlString()+ " " +
fileId).append(NL);

  // tout comme pour 'rankscore', creez le fichier de nom
        //      outputPath + "/download_"+ fileId +".txt"
        // sauvegarder dedans docContent
        // faire flush() et close()

  // sauvegardez le titre dans son fichier, similairement

  // sauvegardez l'url.getUrlString() dans son fichier, similairement

    }
```

9. Finissez le corps de la méthode `SimpleCrawler.saveLinks(CrawlerUrl url)`

```
    private void saveLinks(CrawlerUrl url) throws Exception {
  Collection<String> urlStrings = url . getLinks();
  // TRAVAIL A FAIRE

  // mettez l'indice du document courant (variable numberItemsSaved)
        // comme String dans fileId

  // creez le fichier nomme
  //
  //    outputPath + "/outlinkurls_"+ fileId +".txt"
  //

        // sauvegardez dedans iterativement chaque URL de la
collection
        // urlStrings, suivie de NL

  // flusher et fermer l'ecriture
    }
```

10. Finissez le corps de la méthode `SimpleCrawler.addUrlsToUrlQueue(CrawlerUrl url)`

```
    private void addUrlsToUrlQueue(CrawlerUrl url) {
  // TRAVAIL A FAIRE

  // recuperer la collection d'URL a l'aide d'une methode
  // de la classe CrawlerUrl

        int depth = url.getDepth() + 1;

  // pour chaque chaine de caracteres URL de la collection
        //
  ///// si la liste (table de hachage) des URL visites (visitedUrls)
  /////    ne contient pas cette URL
  /////
        ////////// rajouter a la queue des URL a visiter (methode
Queue.add())
  ////////// un nouvel objet de type CrawlerUrl. Le constructeur
prend
  ////////// comme arguments cette chaine URL et la variable depth
  /////
  ///// finsi
        //
    }
```

- Compilez en lançant tout simplement `make` et vérifiez que le `crawler.jar` a bien été créé.

3. Faire tourner le petit web crawler

Travail à faire

- Regardez brièvement dans le répertoire `data` (qui contient les pages HTML qui feront l'objet du mini-crawling) : examinez `indexClean.html`, qui contient donc des `href` vers un nombre de ces pages, et puis regardez par exemple dans `usa-02.html` -- vous allez voir qu'à la fin cette page renvoie vers `usa-03.html` et `usa-04.html`, mais celles-ci ne sont pas présentes dans `indexClean.html`. Néanmoins, le crawler devra les télécharger également.
- Assurez-vous que le serveur tourne toujours (ou redémarrez-le depuis un autre shell)
- Faites `mkdir test1Tp1`
- Lancez le crawler avec la commande

```
java -cp . -jar crawler.jar 8123 indexClean.html test1Tp1
```

- Vous devez observer comment le crawler télécharge les documents un par un, explorant ainsi le petit graphe des liens.
- Allant dans le sous-répertoire `test1Tp1`, vous allez retrouver dans le fichier `crawl.txt` les URL et les numéros d'identification des documents chargés. Regardez dedans, pour vous convaincre que le texte a bien été sauvegardé
- Parmi ces fichiers il y en a un qui est vide -- à quoi correspond-il ? Est-ce une erreur du crawler ou bien simplement quelque chose causé par les hyperliens ? Vérifiez votre réponse en allant dans le répertoire `data` et en regardant directement dans les fichiers .html que le serveur a fourni au crawler.
- Regardez maintenant le contenu du fichier `robots.txt` et vérifiez que malgré la présence du lien vers ce fichier depuis `indexClean.html`, le crawler s'est bien comporté et ne l'a pas chargé
- Une première direction possible d'extension du crawler est celle lui permettant de traiter d'autres documents que seulement ceux de type HTML. Regardez brièvement sur le web pour trouver des packages Java pour analyser des PDF, PostScript, MSWord, etc. Quelle étape faudrait insérer dans l'algorithme de `SimpleCrawler.crawl()` pour traiter ces autres types de documents.
- Que pourrions-nous rajouter comme pas dans la filtration des URLs nouvellement découvertes (dans chaque document fraîchement chargé) avant de les rajouter dans la queue ? Pensez qu'on voudrait "cibler" un peu le crawling, en fonction notamment du contenu (donc texte) découvert au fur et à mesure.

4. Liens sortants vers nos autres documents

Pour la suite des TPs il sera nécessaire d'avoir la liste des ids des documents vers lequel un document donné pointe avec ses liens. Autrement dit, si vous revenez regarder brièvement le code, la méthode `SimpleCrawler.saveLinks()` génère les fichiers de la forme `outlinkurls_2.txt`. Regardez dans un de ces fichiers -- vous allez voir des URLs. Ce qu'on veut par la suite c'est non pas ces URLs de manière explicite, mais simplement les numéros d'id de chaque document correspondant.

Travail à faire

- Regardez dans le fichier `biz-01.html` du groupe de fichiers crawlés (attention, il faut regarder dans `data` pour le html initial, car le `download_1.txt` ne convient pas), pour récupérer à la main les URLs des `href` -- liens extérieurs. Écrivez sur une feuille de papier les noms de ces fichiers vers lesquels renvoie le document 1.
- Regardez dans le fichier `outlinkurls_1.txt` du répertoire `petitExemple` et comparez son contenu
- Cherchez maintenant les numéros d'identification de ces autres documents (dans le fichier `crawl.txt`).
- Dessinez sur une feuille de papier des points, représentant les trois documents, numérotés 1,2 et 3, et des flèches pour représenter les liens sortant du 1 vers 2 et 3.
- Regardez dans le fichier `outlinks_1.txt` du répertoire `petitExemple` et écrivez sur la feuille de papier son contenu. Comparez-le avec vos résultats.
- La méthode `SimpleCrawler.generateOutlinksIds()` génère les fichiers `outlinks_*.txt` à partir de leurs fichiers `outlinkurls_*.txt` et de la correspondance donnée dans `crawl.txt`. Écrivez cette méthode.

```
    private void generateOutlinksIds() throws Exception{
       BufferedWriter nTotDocOutput =
           new BufferedWriter(
               new FileWriter(outputPath + "/nTotDocs.txt"));
       nTotDocOutput . write(String . valueOf(numberItemsSaved));
       nTotDocOutput . flush();
       nTotDocOutput . close();
       System.out.println("Enfin: mapping des outlinkurls dans des
outlink"+
                          " ids, pour ecrire les fichiers outlink_");
       // TRAVAIL A FAIRE

       // declarez un objet nomme urlToIdMap de la classe Map, et
       // instanciez-le avec une nouvelle HashMap<String,String,>

       // creez un nouveau Scanner d'un nouveau FileInputStream pour
       // lire le fichier nomme
       //        outputPath + "/crawl.txt"
       //
       // (un exemple de code de lecture de fichier se trouve dans
       // server/SimpleHttpHandler.java)

       // pour chaque ligne lue depuis ce fichier
       //////
       ////// separer la ligne a l'aide de String.split("\\s+")
       //////
       ////// mettez les deux morceaux en tant que nouvelle entree dans
       ////// urlToIdMap (methode .put())
       //
       // finboucle

       System.out.println("J'ai lu tout le crawl.txt, et je "+
                          "vais ecrire les outlink_ files");

       Iterator pDoc=urlToIdMap.entrySet().iterator();

       // on parcours dans une boucle la table de hachage urlToIdMap
       // a l'aide de iterateur pDoc :pDoc.next();
       ////
       //// recuperer l'entree courante  Map.Entry docEntry
       //// se prendre un objet de type StringBuilder
```

```
        ////
         //// ouvrir en lecture le fichier de liens nomme
        ////      outputPath + "/outlinkurls_" + docId + ".txt"
        //// (avec un nouveau Scanner, etc. comme plus haut)
        ////
        ////
        //// pour chacune de ses lignes (dans une boucle),
        ////    obtenir l'id du document (docId depuis urlToIdMap (s'il est
dedans)
        ////    rajouter cet id a notre objet StringBuilder a l'aide
        ////    de la methode append()
        ////
        //// finboucle
        ////
        //// ecrire le .toString() de notre objet StringBuilder dans
        //// le fichier de liens par id nomme
        ////
        ////     outputPath + "/outlinks_"+ docId +".txt"
        ////
        // finBoucle
    }
```

5. Documentation

En général en Java on peut mettre des commentaires dans le code suivant une syntaxe particulière, et alors l'outil javadoc peut générer automatiquement des pages html avec la documentation de notre code. Ceci est souvent utile lorsqu'on veut expliquer comment se servir d'une nouvelle API, etc.

Travail à faire :

Mettez les commentaires nécessaires pour javadoc dans le code du crawler, faites tourner javadoc dessus et regardez ensuite les pages web ainsi générées (en utilisant le serveur simple http de ce TP).