Université de Technologie de Compiègne

Génie Informatique

Rapport de TD

LO17

Steve LAGACHE & Romain PELLERIN

Chargé de TD : Pierre Morizet-Mahoudeaux

Printemps 2016 (P16)

Dernière mise à jour : 13 avril 2016

Table des matières

1 TD1 : Génération XML

L'objectif de ce TD était de s'initier au "*parsing*" en Perl et d'être capable d'extraire certaines informations (uniques ou non) d'un seul document d'abord, puis ensuite de plusieurs. Nous avions un peu plus de 326 pages HTML contenant toutes un article issu d'un même site scientifique. Il nous fallait extraire des données "uniques" comme le numéro de l'article ou sa rubrique, et des données présentes une (parfois zéro) ou plusieurs fois, comme les paragraphes du texte principal ou les images présentes.

1.1 Méthodologie employée

Pour la totalité des éléments à récupérer, nous avons utilisé des *regex* (sauf pour le nom du fichier que nous récupérerions dans `$ARGV`).

Pour produire notre XML, nous avons décidé de tout afficher sur la sortie standard. Il s'agira ensuite de rédiriger cette sortie standard dans un fichier grâce à l'opérateur `>` d'Unix. Cela est bien plus simple que manipuler l'écriture de fichier(s) en Perl.

1.1.1 Repérer un élémént d'intérêt

Pour chaque élément unique qu'il nous fallait récupérer, nous avons essayé de trouver la structure de balises HTML avoisinante qui soit unique. Pour cela, nous avons beaucoup fait appel à la structure `.*?` (n'importe quel caractère zéro, une ou plusieurs fois) dans son mode *lazy*. Cela nous a permis de simplifier nos *regex*. Voyons quelques exemples.
Pour récupérer le numéro, la structure HTML autour était suffisamment simple pour ne pas avoir besoin d'utiliser `.*?` :

```
$body=~/<span class="style95" style="color:inherit">(\d+)<\/span
   ><\/a>/;
```

En revanche, nous en avons eu besoin pour la partie contact :

```
$body =~ /Pour en savoir plus, contacts :.*?<p class="style44"><
   span class="style85">(.*?)<\/span>/s;
```

L'utilisation de parenthèses nous permet d'identifier le ou les groupes d'intérêt.

1.1.2 Lecture du contenu d'un fichier

Nous avons été capables de récupérer le contenu d'un fichier donné en argument grâce à une boucle sur l'opérateur "diamant" (`<>`). Fondalement, cet opérateur permet de boucler sur chaque ligne de l'*input*. Nous avons donc simplement récupéré chaque ligne du fichier et avons concaténé ces lignes à une variable `$body` initialisée à une chaîne de caractères vide.

1.1.3 Lecture du contenu de plusieurs fichiers

Pour pouvoir récupérer toutes les lignes de chaque fichier tout en dissociant les fichiers, nous avons du opter pour une technique légèrement différente :

1. Nous créons une tableau.

2. Chaque "case" du tableau sera une variable similaire à `$body` (dont nous avons parlé au-dessus) : elle contiendra toutes les lignes **d'un seul fichier**.
3. Grâce à l'opérateur diamant, nous bouclons sur toutes les lignes de l'*input*.
4. Nous sommes capable de savoir à tout instant quel fichier nous sommes en train de lire grâce à la variable `$ARGV`.
5. Une fois que nous avons récupéré tous les différents documents dans chaque case du tableau, nous commençons à boucler sur ce tableau de la même manière que nous le faisions pour un seul fichier.

```
@htmls;
while (<>) {
  $fichier = $ARGV;
  $fichier=~s/.*\///g;
  if (!defined(@htmls{$fichier})) {
    $htmls{$fichier} = $_;
  }
  else {
    $htmls{$fichier} .= $_;
  }
}

print "<corpus>\n";
while (($fichier,$html) = each(%htmls)) {
  print "<bulletin>\n";
  ...
  print "</bulletin>\n";
  }
print "</corpus>\n";
```

À noter que nous n'avons pas utilisé la fonction `chop` pour les lignes de l'*input*. Cela n'a pas beaucoup d'incidence, il nous faudra seulement penser dans nos futures regex à prendre en compte le caractère `\n` (souvent en utilisant l'option `/s`).

1.2 Commandes Unix

1.2.1 Génération du XML

Voici la commande Unix utilisée pour récupérer la sortie standard de notre programme et la rediriger dans un fichier XML. Il faut également préciser que nous avons rendu notre programme exécutable grâce à `chmod`. De plus, nous utilisons le script `convert.pl` pour convertir les entités HTML en caractères Unicode.

```
./td1.pl BULLETINS/*.htm | perl convert.pl > output.xml
```

1.2.2 Vérification du nombre d'images

Nous avons également utilisé quelques commandes Unix supplémentaires dans le but de vérifier que nous récupérerions bien le nombre exactes d'images présentes dans les articles. Par exemple, nous avons utilisé `grep` dans son mode *regex*.

```
./td1.pl BULLETINS/*.htm | perl convert.pl > output.xml && { grep "
   <image>" output.xml } | wc -l && echo "Images parsées en Perl"
   && { grep -aoE "streaming.+?\.jpg" BULLETINS/*.htm && grep -aoE
   "<img.*?www\.bulletins-electroniques\.com\/Resources_fm\/
   actualites.*?\.jpg" BULLETINS/*.htm } | wc -l && echo "Images
   dans les fichiers d'origine";
```

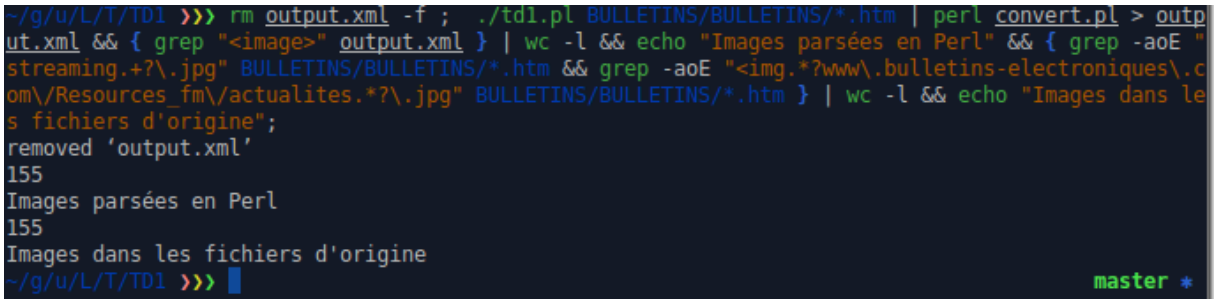

Figure 1.1 – Résultat de la commande

Ce n'est cependant pas la meilleure technique puisque nous réutilisons les mêmes *regex* que celles utilisées dans notre script Perl.
Une meilleure solution consisterait à compter toutes les balises image dans les fichiers et à soustraire le nombre d'images qui ne font pas partie des articles (celles sur les côtés par exemple).

1.3 Code source

L'intégralité du code source de ce TD1, commenté, est trouvable en annexes.

2 TD2 : Indexation

Le but de ce second TD est d'utiliser le fichier XML que nous avons produit lors du TD1 afin de créer des index (des fichiers inverses). Pour cela, nous avons à notre disposition plusieurs scripts Perl.

1. **segmente_TT.pl** : permet, à partir d'un fichier XML, d'extraire tous les mots individuellement contenus entre les balises `<titre>` ou `<texte>`.
2. **newcreeFiltre.pl** : permet de créer un script Perl à partir d'un fichier à une ou deux colonnes : ce script généré remplacera les mots de la première colonne par ceux de la seconde (ou par rien s'il n'y a qu'une colonne) dans un fichier donné en paramètre à ce script.
3. **successeurs.pl** et **filtronc.pl** ; ces deux scripts s'utilisent de pair, afin de :
   (a) Générer la liste des succésseurs pour chaque lettre des mots d'une liste de mots.
   (b) Créer à partir des résultats obtenus à un fichier à deux colonnes associant un mot à un lemme.
4. **index.pl** : permet de créer à partir d'un corpus (XML) un fichier inverse sur une balise donnée en argument
5. **indexText.pl** ; permet de créer un fichier inverse à partir d'un flux de données (entrée standard) de la forme "mot rubrique fichier numéro".

2.1 Obtenir les tf

Grâce au premier script, nous sommes capable d'obtenir la liste des mots contenus dans les textes et titres de tous les documents HTML. Puis, à l'aide des commandes Unix telles que `sort` et `uniq` (avec l'argument `-c`), nous obtenons pour chaque mot son **tf** (nombre d'occurences).

```
cat ../TD1/output.xml| ./segmente_TT.pl -f | sort | uniq -c | sed -
   e 's/^\s*//' > tf.txt
```

La commande `sed` nous permet de supprimer les espaces inutiles en début de chaîne, qui sont automatiquement générés par les commandes.

2.2 Obtenir les df

Ensuite, pour obtenir les **idf** (nombre de fichiers dans lesquel un mot apparaît), nous avons deux possibilités : réutiliser le fichier **tf.txt** directement ou repartir du fichier XML produit lors du TD1. Voici comment faire à partir du fichier **tf.txt** :

```
cat tf.txt | cut -d' ' -f2 | cut -f1 | sort | uniq -c | sed -e 's
   /^\s*//' > df.txt
```

Ici à nouveau nous retirons les espaces en début de chaine. La commande `cut` nous permet de sélectionner uniquement certaines "colonnes" de notre fichier source.

2.3 idf

Ensuite, nous devons calculer les **idf** des mots. Cela implique notamment de calculer un logarithme 10. Nous avons donc créer un script Perl qui prend en argument notre fichier texte contenant les df. Le sortie standard de ce script est redirigée vers un fichier `idf.txt`.

```
sub log10 {
    my $n = shift;
    return log($n)/log(10);
}

while (<>) {
    /(\d+)\s+(.*)/;
    print $2."\t".log10(326/$1)."\n";
}
```

2.4 tf*idf

La dernière étape avant de passer aux stop-lists est de caculer le quotient **tf*idf**. Cela se fait très facilement grâce à un script Perl utilisant un tableau associatif. Le script traite d'abord un fichier texte contenant les idf, en les stockant dans ce tableau. Ensuite, le script traire le fichier concernant les tf : c'est la que la multiplication est faite et que le résultat est affiché sur la sortie standard (que l'on va bien sûr rediriger vers un fichier).

```
# Usage: perl % idf.txt tf.txt

@mots;
while (<>) {
    if ($ARGV =~ m/idf/) {
        /(.*?)\s+(\d+(\.\d+)?)/;
        $mots{$1} = $2;
    }
    elsif ($ARGV =~ m/tf/) {
        /(\d+)\s+(.*?)\s+(.*)/;
        $nb = $1;
        $mot = $2;
        $file = $3;
        $tfidf = $nb * ($mots{$mot});
        print $file."\t".$mot."\t".$tfidf."\n";
    }
}
```

2.5 Stop-liste

Maintenant, il s'agit de générer une stoplist afin de filtrer le fichier XML généré lors du TD1. Nous avons choisi un seuil en dessous duquel les mots seront ajoutés à la stop liste car considérés comme n'apportant rien au document (par exemple des déterminantes, des mots de liaison, etc.). Une script Perl nous permet de faire cela :

```
# Usage: perl % tfidf.txt
```

```
@words;
while (<>) {
    /(.*?)\s(.*?)\s(\d+(\.\d+)?)/;
    if ($3 lt '0.79189260882435') {
        $words{$2} = 1;
    }
}

while (($key,$value) = each(%words)) {
    print $key."\n";
}
```

Puis, à l'aide du script fourni n°2, nous pouvons créer un script Perl qui supprimera automatiquement les mots présents dans notre stop-liste, afin de filtrer le fichier XML.

```
# Create script to remove words
rm -f removeWordsFromXML.pl; ./newcreeFiltre.pl stoplist.txt >
   removeWordsFromXML.pl && chmod +x removeWordsFromXML.pl

# Create new XML without the words from the stop list
rm -f newOutput.xml; ./removeWordsFromXML.pl corpus.xml > newOutput
   .xml
```

3 Conclusion

Appendices

A TD1 : code

Ce code fonctionne avec un ou plusieurs fichiers.

```
#!/usr/bin/perl

use utf8;
binmode(STDOUT, ":utf8");

# HTML AND BODY

@htmls;
while (<>) {
  # FILENAME
  $fichier = $ARGV;
  $fichier=~s/.*\///g;
  s/\n//;
  if (!defined(@htmls{$fichier})) {
    $htmls{$fichier} = $_;
  }
  else {
    $htmls{$fichier} .= $_;
  }
}

print "<corpus>\n";
while (($fichier,$html) = each(%htmls)) {
  print "<bulletin>\n";
  print '<fichier>'.$fichier."</fichier>\n"; # si on donne *.htm en
      param, argv[0] contient le premier, [1] un autre, etc
  $body = $html;
  $body=~/<body.*?>(.*)<\/body>/sg;
  $body = $1;


  # NUMERO

  $body=~/<span class="style32">BE France (\d+).*?<\/span>/;
  $numero = $1;
  print '<numero>'.$numero."</numero>\n";


  # DATE

  $body=~/<span class="style42">.*?(\d{1,2}\/\d{1,2}\/\d{4})<\/span
     >/;
  $date = $1;
  print '<date>'.$date."</date>\n";
```

```
# RUBRIQUE AND TITRE

$body=~/<p class="style96"><span class="style42">(.+)<br><\/span
   ><span class="style17">(.*?)<\/span><\/p>/;
$rubrique = $1;
$titre = $2;
print '<rubrique>'.$rubrique."</rubrique>\n";
print '<titre>'.$titre."</titre>\n";


# TEXTE

$body =~ /<td.*?class="FWExtra2".*?>.*?<span class="style95"
   >(.*?)<td.*?class="FWExtra2"/s;
$texte = $1;
$texte =~ s/<br \/>\n?<\/span><div style="text-align: center"><
   img.*?class="style95"><br \/>//gs;
$texte =~ s/<.*?>//g;
$texte =~ s/^\s+|\s+$|\n//g;
print "<texte>".$texte."</texte>\n";
# (.*?) makes it lazy (cf http://forums.phpfreaks.com/topic
   /265751-how-does-it-work/) instead of greedy


# IMAGES WITH URL

#@array = ($body =~ /(www\.bulletins-electroniques\.com\/
   Resources_fm\/actualites.*?\.jpg)/g);
#print join "--\n", @array;
print "<images>\n";
pos($body) = 0; # reset cursor position cause /g leave the cursor
    in the middle of nowhere
while ($body =~ /(www\.bulletins-electroniques\.com\/Resources_fm
   \/actualites.*?\.jpg).*?<strong>(.*?)<\/strong>|(streaming
   .+?\.jpg)/sg) {
    print "<image>".$1.$3."</image>\n";
    print "<legende>".$2."</legende>\n";
}
print "</images>\n";


# CONTACT

print "<contact>";
$body =~ /Pour en savoir plus, contacts :.*?<p class="style44"><
   span class="style85">(.*?)<\/span>/s;
$contact = $1;
$contact =~ s/<.*?>//gs;
print $contact;
```

```
  print "</contact>\n";
  print "</bulletin>\n";

  if (!defined($fichier) || !defined($numero) || !defined($date) ||
      !defined($rubrique) || !defined($titre) || !defined($texte)
     || !defined($contact)) {
    print STDERR "Missing one field";
    exit -1;
  }
  $fichier = $numero = $date = $rubrique = $titre = $texte =
     $contact = undef;
}
print "</corpus>\n";
```