LO17 – Indexation et recherche d'information

Rapport 1 : Préparation et indexation du corpus

Étudiants :

Joanne ANTONETTI

Tudor LUCHIANCENCO

Professeur :

Pierre MORIZET-MAHOUDEAUX

utc Université de Technologie Compiègne

LO17 – Indexation et recherche d'information

Sommaire

LO17 – Indexation et recherche d'information

LO17 – Indexation et recherche d'information

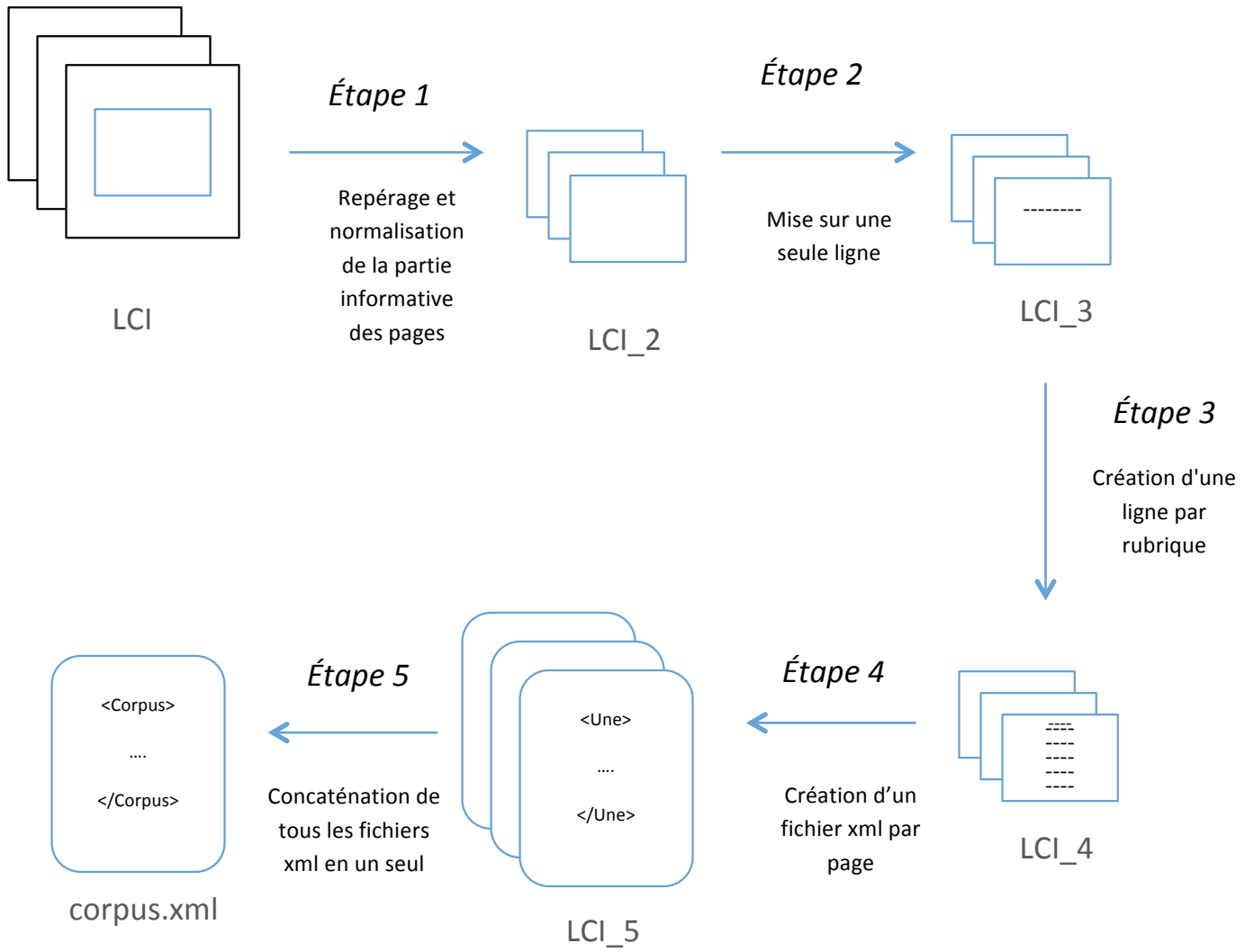
Introduction

Structure d'une page web du site LCI : (exemple avec la page « lci-monde-2005-03-01.html »)

LO17 – Indexation et recherche d'information

Préparation du corpus

Résumé du travail réalisé



LO17 – Indexation et recherche d'information

Travail réalisé

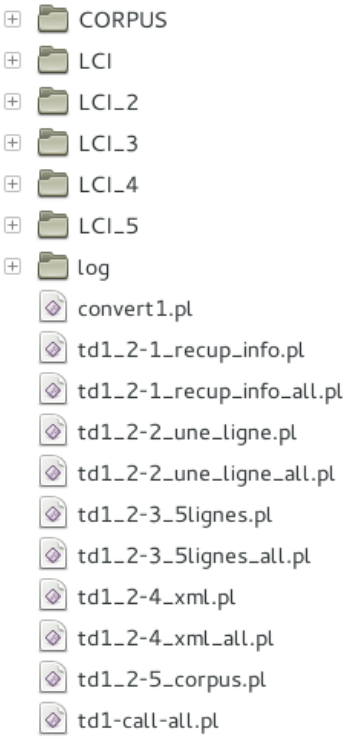
Pour la préparation de notre corpus, nous avons du préparer tout un ensemble de scripts perl.

Notre méthodologie est la suivante : pour une opération (étape) donnée, créer un script perl qui traite un fichier source (html) qui fonctionne. Une fois ceci réalisé, écrire un second script plus léger qui fait appel au premier pour tous les fichiers source qu'on doit traiter.

Le premier script ("le travailleur") prend un fichier source en argument, vérifie sa cohérence par rapport à ce qu'il doit contenir (tests de validation) et procède ensuite au traitement concerné. Les tests de validation affiche des messages d'information/d'erreur sur la sortie standart STDOUT.

Le second script ("le manager") fait appel au travailleur autant de fois qu'il y a des fichiers sources à traiter. Il écrit dans un fichier de log la sortie STDOUT de l'éxécution des travailleurs. Enfin, il vérifie le nombre de fichiers générés dans le dossier de destination.

Organisation de nos fichiers



Etape 1 : Repérage et normalisation de la partie informative des pages LCI-Monde

Dossier source : LCI

Tout d'abord il a fallu extraire la partie informative de l'ensemble des pages LCI que nous avions à traiter. Cette partie contient toutes les informations dont on a besoin pour la construction de notre corpus.

Nous avons détécté que cette partie informative est délimitée par les expressions : "/([^\/]Bloc IBL_ID|Blc)=27303/" pour le début de partie et "/(\/Bloc IBL_ID|\/Blc)=27916/" pour la fin de partie.

Le script "td1_2-1_recup_info.pl" parcours chaque ligne du fichier source et écrit dans un fichier de destination portant le même nom que celui source les lignes trouvées entre les expressions de début et de fin.

Le script normalise aussi chaque ligne écrite dans le fichier de destination au format "iso8859-1".

Résultat de l'étape : dans le dossier LCI_2, un ensemble de fichiers html portant le même nom que les sources, et contenant la partie informative qui nous intéresse.

Etape 2 : Mise sur une seule ligne

Dossier source : LCI_2

Pour mieux exploiter notre partie informative, nous avons du mettre sur une seule ligne le contenu de chaque fichier source.

Le script "td1_2-2_une_ligne.pl" parcours chaque ligne du fichier source et supprime les caractères de fin de ligne, tabulation; etc. et écrit dans un fichier de destination portant le même nom le résultat.

Le traitement appliqué sur chaque ligne est le suivant : $ligne =~ s/(\n|\r|\f)//g

Résultat de l'étape : dans le dossier LCI_3, un ensemble de fichiers html portant le même nom que les sources, et contenant la partie informative sur une seule ligne.

Etape 3 : Création d'une ligne par rubrique (5 lignes)

Dossier source : LCI_3

Nous avons en entrée un ensemble de fichiers html comportant chacun une ligne.

Le but est maintenant de créer les 5 rubriques (lignes) pour chaque fichier, à savoir :

- UNE : /(<!-- (Bloc IBL_ID|Blc)=27914.*?Lire l'article)/
- A VOIR AUSSI : /(A voir aussi :.*<!-- \/(Bloc IBL_ID|Blc)=27914)/
- FOCUS : /(<!-- (Bloc IBL_ID|Blc)=27913.*<!-- \/(Bloc IBL_ID|Blc)=27913)/
- GROS TITRES : /(<!-- (Bloc IBL_ID|Blc)=27915.*<!-- \/(Bloc IBL_ID|Blc)=27915)/
- RAPPELS : /(<!-- (Bloc IBL_ID|Blc)=27916.*<!-- \/(Bloc IBL_ID|Blc)=27916)/

Résultat de l'étape : dans le dossier LCI_4, un ensemble de fichiers html portant le même nom que les sources, et contenant les 5 rubriques.

Etape 4 : Création des fichiers XML

Dossier source : LCI_4

Nous devons maintenant écrire un fichier xml structuré et détaillé à partir d'un fichier contenant les 5 rubriques.

Nous récupérons par exemple à partir du nom du fichier source la date qu'on formate en "dd/mm/yyyy". Nous lisons une ligne pour traiter la partie UNE, et ainsi de suite. Dans la partie UNE par exemple, nous lisons l'urlArticle, etc. et si l'expression regex cherchée n'est pas présente le contenu sera vide : <urlArticle></urlArticle>.

Résultat de l'étape : dans le dossier LCI_5, un fichier possède cette structure :

```
<PAGE_LCI>

     <FICHIER>lci-monde-2005-02-25.html</FICHIER>
    <DATE_PAGE>25/02/2005</DATE_PAGE>
    <UNE>
        <urlArticle>/news/monde/0,,3204466-VU5WX0lEIDUy,00.html</urlArticle>
        <titreArticle>Le pape muet</titreArticle>
        <dateArticle>25/02/2005</dateArticle>
        <urlImage>http://s.tf1.fr/mmdia/i/47/7/2018477_5.jpg</urlImage>
        <resumeArticle> Le pape a subi jeudi soir une trachéotomie pour l'aider à respirer. L'acte chirurgical a
duré 30 minutes. L'opération laisse toutefois le souverain pontife sans voix, ce qui risque de relancer la
question de sa démission.
        </resumeArticle>
        <mailto>dstraus@tf1.fr</mailto>
        <auteur>D.S. (d'après AFP)</auteur>
    </UNE>
    <LES_VOIRAUSSI>
        <VOIRAUSSI>
            <dateArticle>24/02/2005</dateArticle>
            etc.
```

extrait du code montrant la création d'un fichier xml détaillé

```
23    print FICOUT "<PAGE_LCI>\n";
24
25    # nom fichier
26    print FICOUT "<FICHIER>$fichier</FICHIER>\n";
27
28    # date
29    $fichier =~ /(\d{4})-(\d{2})-(\d{2})/;
30    $date_fic = $3."/".$2."/".$1;
31    $date_fic_mois = $2;
32    $date_fic_annee = $1;
33    print FICOUT "<DATE_PAGE>$date_fic</DATE_PAGE>\n";
34
35    # lecture ligne UNE
36    $ligne = <FICIN>;
37    print FICOUT "<UNE>\n";
38        # url
39        print FICOUT "<urlArticle>";
40        if ($ligne =~ /<a href="(.*)" class="S431"/) {
41            $url = $1;
42            print FICOUT $url;
43        }
44        print FICOUT "</urlArticle>\n";
```

Création d'une date valide :

Si on la Une date de janvier et que les autres articles datent de décembre, alors ces articles datent de l'année précédentes (on a changé d'année il y a peu de jours)

```
198  # lecture ligne RAPPEL
199  $ligne = <FICIN>;
200  print FICOUT "<LES_RAPPELS>\n";
201      while ($ligne =~ /(span class="S48">.*?)<\/table>/g) {
202          $rappel = $1;
203          print FICOUT "<RAPPEL>\n";
204              print FICOUT "<dateArticle>";
205              if ($rappel =~ /span class="S48">(\d\d) (.*?)<\/span>/) {
206
207                  #~ %mon2num = qw(
208                    #~ janvier 01  février 02 mars 03  avril 04  mai 05  juin 06
209                    #~ juillet 07  août 08  septembre 09  octobre 10 novembre 11 décembre 12
210                  #~ );
211                  #~ $mois = $mon2num{lc $2};
212
213                  $jour = $1;
214
215                  if ($2 =~ /janvier/) { $m = "01"; }
216                  if ($2 =~ /f.vrier/) { $m = "02"; }
217                  if ($2 =~ /mars/) { $m = "03"; }
218                  if ($2 =~ /avril/) { $m = "04"; }
219                  if ($2 =~ /mai/) { $m = "05"; }
220                  if ($2 =~ /juin/) { $m = "06"; }
221                  if ($2 =~ /juillet/) { $m = "07"; }
222                  if ($2 =~ /ao.t/) { $m = "08"; }
223                  if ($2 =~ /septembre/) { $m = "09"; }
224                  if ($2 =~ /octobre/) { $m = "10"; }
225                  if ($2 =~ /novembre/) { $m = "11"; }
226                  if ($2 =~ /d.cembre/) { $m = "12"; }
227
228                  $mois = $m;
229                  $an = $date_fic_annee;
230
231                  # si date rappel = décembre et date fic = janvier
232                  if ($mois == "12" && $date_fic_mois == "01") {
233                      $an = $date_fic_annee - 1;
234                  }
235
236                  $date_finale = $jour."/".$mois."/".$an;
237                  print FICOUT $date_finale;
238              }
239              print FICOUT "</dateArticle>\n";
```

Etape 5 : Création du fichier XML final (Création du corpus)

Dossier source : LCI_5

Cette étape est la plus simple du processus de préparation du corpus.

Il s'agit de la concaténation de tous les fichiers présents dans LCI_5, les fichiers xml détaillés.

Résultat de l'étape : dans le dossier CORPUS (corpus.xml), le corpus qui contient la concaténation de tous les fichiers xml détaillés de LCI_5 avec les balises <CORPUS></CORPUS> en plus.

Tests de vérification

Pour chaque étages

On généré un fichier de logs. On a donc générés les 5 fichiers suivants :

étape	Fichier du log
1	log_recup_info
2	log_une_ligne
3	Log_5lignes
4	Log_xml
5	log_corpus

A chaque fois, on vérifie le nombre de fichiers générés.

```
25  ##########################################
26  # test nombre fichiers traités
27  ##########################################
28  $nb_fic_out = `ls LCI_2 | wc -l`;
29  print FICLOG "Nombre de fichiers traités : $i\n";
30  print FICLOG "Nombre de fichiers dans le dossier LCI_2 d'après commande Unix (wc -l) : $nb_fic_out";
```

Voici le log de l'étape 1 :

```
# tudor@tudor-Latitude-E6520  ~/Documents/UTC_Genie_Informatique/UTC_5A_GI06_SRI/LO17/LO17/TD1/log
> cat log_recup_info
Nombre de fichiers originaux : 328
Normalisation de fichiers au format iso8859-1
Nombre de fichiers traités : 328
Nombre de fichiers dans le dossier LCI_2 d'après commande Unix (wc -l) : 328
```

Etape 1 : Repérage et normalisation de la partie informative des pages LCI-Monde

On test l'existence et l'unicité des balises délimitant la partie informative.

```
21  ##########################################
22  # test existence et unicité
23  ##########################################
24  open(FICIN, "LCI/$fichier") || die ("Erreur d'ouverture du fichier");
25  $t_deb_info = 0;
26  $t_fin_info = 0;
27  while ($ligne = <FICIN>) {
28      if ($ligne =~ /([^\/]Bloc IBL_ID|Blc)=27303/) {
29          $t_deb_info++;
30      } elsif ($ligne =~ /(\/Bloc IBL_ID|\/Blc)=27916/) {
31          $t_fin_info++;
32      }
33  }
34  $t_err = 0;
35  if ($t_deb_info > 1) {
36      print "$fichier : Element début partie informative trouvé $deb_info fois dans le texte\n";
37      $t_err = 1;
38  } elsif ($t_deb_info < 1) {
39      print "$fichier : Element début partie informative NON trouvé dans le texte\n";
40      $t_err = 1;
41  }
42  if ($t_fin_info > 1) {
43      print "$fichier : Element fin partie informative trouvé $fin_info fois dans le texte\n";
44      $t_err = 1;
45  } elsif ($t_fin_info < 1) {
46      print "$fichier : Element fin partie informative NON trouvé dans le texte\n";
47      $t_err = 1;
48  }
49  # erreur
50  if ($t_err > 0) {
51      print "$fichier : Arrêt programme à cause d'erreur, voir fichier log\n";
52      exit;
53  }
54  close FICIN;
```

Résultat :

```
# tudor@tudor-Latitude-E6520  ~/Documents/UTC_Genie_Informatique/UTC_5A_GI06_SRI/LO17/LO17/TD1/log
> cat log_recup_info
Nombre de fichiers originaux : 328
Normalisation de fichiers au format iso8859-1
Nombre de fichiers traités : 328
Nombre de fichiers dans le dossier LCI_2 d'après commande Unix (wc -l) : 328
```

Etape 2 : Mise sur une seule ligne

On test que l'on a bien une seule ligne par fichier.

```
17  ##########################################
18  # test une seule ligne fichier entrée
19  ##########################################
20  open(FICIN, "LCI_3/$fichier") || die ("Erreur d'ouverture du fichier");
21  $i = 0;
22  while ($ligne = <FICIN>) {
23      $i++;
24  }
25  if ($i > 1 | $i < 1) {
26      print "$fichier : Le fichier d'entrée comporte 0 ou plus d'une ligne\n";
27      print "$fichier : Arrêt programme à cause d'erreur, voir fichier log\n";
28      exit;
29  }
30  close FICIN;
```

Etape 3 : Création d'une ligne par rubrique (5 lignes)

On test l'existence et l'unicité de chacune des 5 rubriques.

```
32  ##########################################
33  # test existence et unicité rubriques
34  ##########################################
35  open(FICIN, "LCI_3/$fichier") || die ("Erreur d'ouverture du fichier");
36
37  $t_une = 0;
38  $t_avoiraussi = 0;
39  $t_focus = 0;
40  $t_grostitre = 0;
41  $t_rappel = 0;
42
43  $ligne = <FICIN>;
44  while ($ligne =~ /(<!-- (Bloc IBL_ID|Blc)=27914.*?Lire l'article)/g) {
45      $t_une++;
46  }
47
48  while ($ligne =~ /(A voir aussi :.*<!-- \/(Bloc IBL_ID|Blc)=27914)/g) {
49      $t_avoiraussi++;
50  }
51
52  while ($ligne =~ /(<!-- (Bloc IBL_ID|Blc)=27913.*<!-- \/(Bloc IBL_ID|Blc)=27913)/g) {
53      $t_focus++;
54  }
55
56  while ($ligne =~ /(<!-- (Bloc IBL_ID|Blc)=27915.*<!-- \/(Bloc IBL_ID|Blc)=27915)/g) {
57      $t_grostitre++;
58  }
59
60  while ($ligne =~ /(<!-- (Bloc IBL_ID|Blc)=27916.*<!-- \/(Bloc IBL_ID|Blc)=27916)/g) {
61      $t_rappel++;
62  }
63
64  if ($t_une > 1) {
65      print "$fichier : Element Une trouvé $t_une fois dans le texte\n";
66  } elsif ($t_une < 1) {
67      # verification fichier origine
68      open(FICORIGINE, "LCI/$fichier") || die ("Erreur d'ouverture du fichier");
69      $ligne = <FICORIGINE>;
70      $nb = 0;
71      while ($ligne = <FICORIGINE>) {
72          if ($ligne =~ /(<!-- (Bloc IBL_ID|Blc)=27914.*?Lire l'article)/) {
73              $nb++;
74          }
75      }
76      if ($nb > 0) {
77          print "$fichier : Element Une NON trouvé dans le texte alors qu'il existe dans le fichier d'origine.\n";
78      } else {
79          print "$fichier : Element Une NON trouvé ni dans le texte ni dans le fichier d'origine.\n";
80      }
81      close FICORIGINE;
82  }
83  if ($t_avoiraussi > 1) {
84      print "$fichier : Element Avoiraussi trouvé $t_avoiraussi fois dans le texte\n";
85  } elsif ($t_avoiraussi < 1) {
```

```
# tudor@tudor-Latitude-E6520  ~/Documents/UTC_Genie_Informatique/UTC_5A_GI06_SRI/LO17/LO17/TD1/log
> cat log_5lignes
Nombre de fichiers originaux : 328
lci-monde-2005-02-28.html : Element Avoiraussi NON trouvé ni dans le texte ni dans le fichier d'origine.
lci-monde-2005-03-04.html : Element Avoiraussi NON trouvé ni dans le texte ni dans le fichier d'origine.
lci-monde-2005-03-13.html : Element Avoiraussi NON trouvé ni dans le texte ni dans le fichier d'origine.
lci-monde-2005-03-15.html : Element Avoiraussi NON trouvé ni dans le texte ni dans le fichier d'origine.
lci-monde-2005-03-16.html : Element Avoiraussi NON trouvé ni dans le texte ni dans le fichier d'origine.
lci-monde-2005-03-18.html : Element Avoiraussi NON trouvé ni dans le texte ni dans le fichier d'origine.
lci-monde-2005-03-24.html : Element Avoiraussi NON trouvé ni dans le texte ni dans le fichier d'origine.
lci-monde-2005-03-25.html : Element Avoiraussi NON trouvé ni dans le texte ni dans le fichier d'origine.
lci-monde-2005-03-30.html : Element Avoiraussi NON trouvé ni dans le texte ni dans le fichier d'origine.
```

Etape 4 : Création des fichiers XML

On vérifie que l'url et titre de la Une de chaque page existe et est unique.

```
31  ##########################################
32  # test existence et unicité url partie Une
33  ##########################################
34  my %hash_url = ();
35  # construct hashmap url fichiers
36  foreach $fic (@list) {
37      open(FICTEST, "LCI_5/".$fic) || die ("Erreur d'ouverture du fichier");
38      while ($ligne = <FICTEST>) {
39          if ($ligne =~ /<UNE>/) {
40              # première ligne après UNE
41              $ligne = <FICTEST>;
42              $ligne =~ /<urlArticle>(.*?)<\/urlArticle>/;
43              $url = $1;
44              @valeurs = values(%hash_url);
45              for $elt (@valeurs) {
46                  if ($elt == $url && !elt) {
47                      chomp($fic);
48                      print FICLOG "$fic : url $url déjà trouvé dans un autre fichier.\n";
49                  }
50              }
51              $hash_url{$fic} = $url;
52          }
53      }
54      close FICTEST;
55  }
```

```
# tudor@tudor-Latitude-E6520  ~/Documents/UTC_Genie_Informatique/UTC_5A_GI06_SRI/LO17/LO17/TD1/log
> cat log_xml
Nombre de fichiers originaux : 328
Nombre de fichiers traités : 328
Nombre de fichiers dans le dossier LCI_5 d'après commande Unix (wc -l) : 328

Fichiers possédant une url : 328/328
Fichiers possédant un titre : 328/328
```

Etape 5 : Création du fichier XML final (corpus)

On liste les pages qui ne contiennent pas d'url.

```
84  ##############################################
85  # statistiques
86  ##############################################
87
88  # liste des pages qui n'ont pas de URL
89  @liste_fic = keys(%hash_url);
90  $i = 0;
91  foreach $fic (@liste_fic) {
92      if (!$hash_url{$fic}) {
93          $i++;
94          chomp($fic);
95          print FICLOG "$fic : n'a pas de url.\n";
96      }
97  }
98  $possede_url = $nb_fic_gen - $i;
99  print FICLOG "Fichiers possédant une url : $possede_url/$nb_fic_gen\n";
```

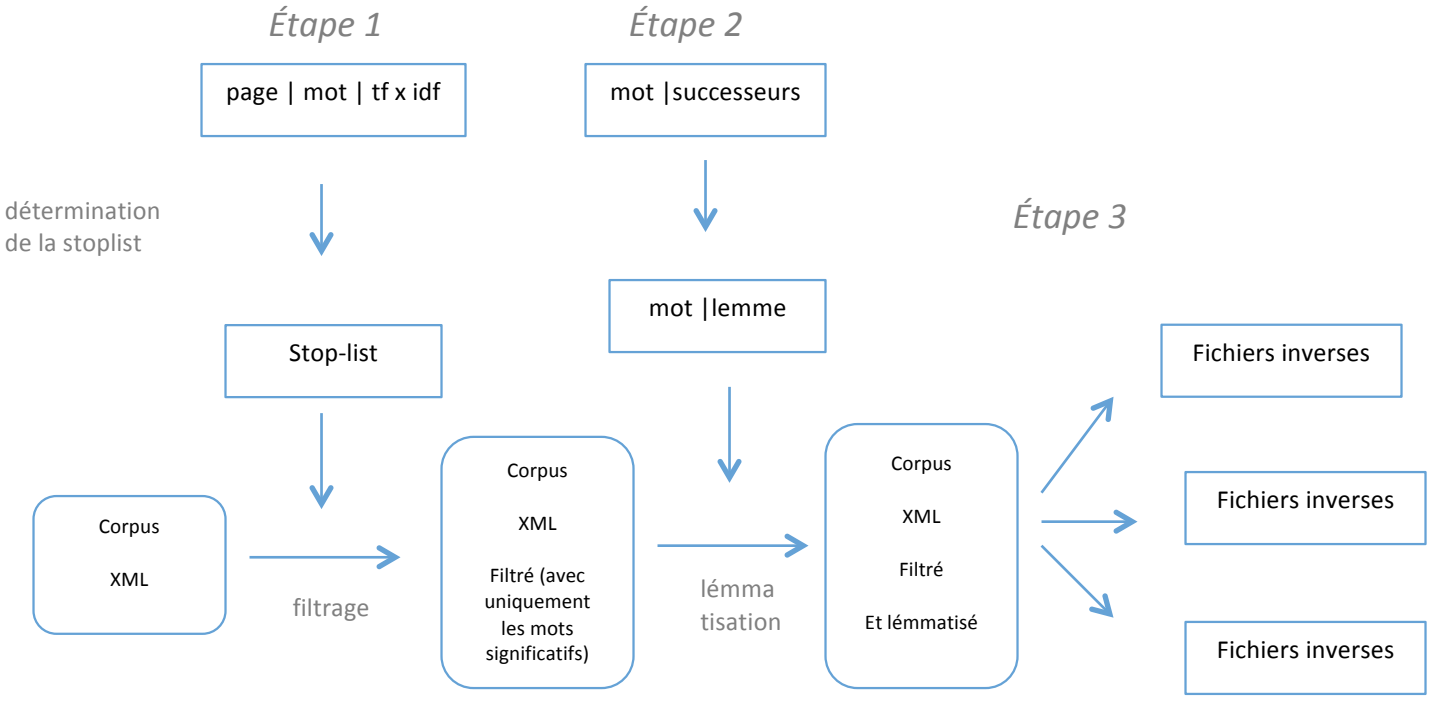
Remarque :

Une fois le fichier xml généré on l'ouvre avec un navigateur web pour vérifier sa structure. On pourrait aussi définir sa DTD pour s'assurer d'une structure correcte.

LO17 – Indexation et recherche d'information

Indexation du corpus

Résumé du travail réalisé



A la fin de l'étape 1 le corpus est débarassés des mots-non significatifs grâce au filtrage avec les mots de la **stop-list**. Cette stop-list ayant été générée grâce au calcul des ***tfxifd*** de chaque mot de chaque page. Le *tf* représentant l'importance d'un mot dans une page et le *idf* indique le pouvoir de discrimination du mot dans le corpus. Ainsi un *tfxifd* petit indique un mot non-significatif.

A la fin de l'étape 2, on a un corpus lemmatisé. En effet, on a remplace chaque mot par son **lemme** (sa classe représentante). Cela permet donc de retrouver une forme dérivée d'un mot, et donc retrouver l'ensemble des pages, titre ou résumé qui contient la même information mais sous une forme différente. Pour déterminer le lemme d'un mot on a d'abord calculer pour chaque lettre du mot le nombre de successeurs possibles du mot, à partir de ces données on a ensuite tronqués le mot pour obtenir le lemme.

A la fin de l'étape 3, on a crée deux types de **fichiers inverses :**

- Indexation des **flux de données** : 1 fichier permettant l'indexer des mots **entre** 2 balises (dans titre et résumé)
- 7 fichiers permettant l'indexation du contenu complet entre les balises **clés** (`"titreArticle"`,

"dateArticle", "urlImage", "resumeArticle", "mailto", "auteur", "themeArticle"), donc en d'autres termes l'indexation des auteurs, des titres, des résumés, etc.

Dans les deux cas, on fait correspondre à chaque mot la liste des fichiers qui le contient. Ces fichiers sont décrits pas : la page web, la rubrique (Une ou GrosTitre ect.) et l'url de l'article

Analyse et test du travail réalisé

Choix de l'unité documentaire

On a choisit comme unité documentaire une page LCI puisque les données contenues dans un article ne sont pas suffisantes pour une exploitation optimale. En d'autres mots, le zoom sur un article est bien trop grand (inexploitable) par rapport à celui sur une page LCI.

Si on choisit l'article comme unité documentaire, presque tous les mots d'un article auront la même fréquence (puisqu'il y aura peu de répétition). Comme le score de correspondance d'un couple document-requête est dépendant de cette fréquence, il sera alors plus difficile de discriminer les documents correspondant à une requête.

Détermination de la règle d'extraction des mots de la stop-list.

Nous avons parcouru la liste des mots classés par ordre croissant de tfxidf. Nous avons remarqué qu'à partir d'un tf*idf = 0.413116254843684, on voit apparaître pour la première fois des mots significatifs. Ainsi tous les mots dont le tf*idf est inférieur à cette valeur seuil seront dans notre stop-list.

Bien sûre après ce seuil on trouve des mots qui sont non significatifs et devraient être dans la stop-list. On remarque que ces mots sont souvent courts. Ainsi on pourrait proposer de mettre dans la stop-list les mots de 1 à 2 lettres. Même si cela impliquerait de perdre des mots significatifs tel que (al : nom propre). Cependant on pourrait imaginer que les mots significatifs de 2 lettres sont des noms propres et ont ainsi une majuscule. Il faudrait donc prendre cela en compte et redéfinir les différentes étapes en fonction.

On aurait aussi faire un petit travail statistique complément et calculer pour chaque mot la moyenne et l'écart type de son tf*idf, et déterminer notre stop-list avec ses mesures.

Test de vérification du filtrage du corpus avec la stop-list

Affiche à l'écran les lignes (de titre article ou résumé article) qui contiennent des mots de la stop-list.

```perl
#!/usr/bin/perl

################################
# Tests de vérification stopliste
################################

open(FICHSTOP, "res/stopliste_chiff.txt") || die ("Erreur d'ouverture du fichier");

while ($ligne = <FICHSTOP>) {
    $ligne =~ /(.*?)\n/;
    print `cat CORPUS/corpus_stopliste.xml | grep '<titreArticle>\|<resumeArticle>' | grep -e '\s$1\s'`
}

```

Avec ce test aucune ligne n'est affichée, ce qui montre que le corpus a bien été filtré.

Commentaire des résultats de la lemmatisation

Montrons d'abord un exemple de bonne lemmatisation de l'algorithme. Le lemme « laiss » qui a permit de regrouper un ensemble des mots de sa famille : *laissaient, laissant, laisse, laisser, laissera, laisseront, laissé, laissée, laissées*

Nous remarquons que dans certains cas le lemme est trop grand. En effet, il inclus une partie de la terminaison du mot. Cela est non seulement inutile mais faux. En effet, cela ne permet plus de regrouper un ensemble de forme dérivée sous une forme commune. C'est le cas du lemme « grand » qui aurait du être trouvé pour les mots suivants : *grand, grande, grandes, grandi, grandit, grands.* Dans ce cas le lemme du mot est le mot lui-même au lieu du radical « grand ».

On retrouve le même problème de lemme trop grand avec la famille de mot suivante :

mot	lemme
accusateur	accusat
accusation	accusation
accusations	accusation
accuse	accuse
accusent	accusent
accusé	accusé
accusée	accusée
accusées	accusées
accusés	accusés

On remarque que l'algorithme de lemmatisation n'a pas très bien fonctionné. En effet, ces mots ayant un sens commun, ils aurait du avoir un radical commun tel que « accus ».

A l'inverse, on remarque quand dans certains cas le lemme est trop court et englobe trop de mots au sens différents. C'est le cas de l'exemple ci-dessous. Il y a des la perte d'information, puisque le lemme « écla » ne permet pas de distinguer les mots de famille de éclabousser et le mot de la famille de éclat et éclair.

mot	lemme
éclabousse	écla
éclaboussé	écla
éclaboussée	écla
éclair	écla
éclaircissements	écla
éclat	écla
éclate	écla
éclaté	écla

On constate donc la difficulté de trouver pour tous les mots, le lemme de la bonne taille ni trop court (regroupant trop de mots) ni trop long (regroupant pas assez de mot).

Test sur la création des fichiers inverses

Afin de tester le bon fonctionnement de la création des tables inverses, nous avons vérifié que des mots comme "pays" ou des noms d'auteurs se trouvent bien dans les rubriques et fichiers html indiqués. Les tests ont toujours été positifs.

Conclusion

A l'aide de script PERL et à partir d'un ensemble de pages web (HTML), nous avons créer un corpus et son indexation. Pour ce faire nous avons successivement :

- Extrait les données qui nous intéressaient
- Généré un corpus structuré (XML)
- Filtré le corpus afin de retirer les mots non porteurs d'information
- Lemmatisé le corpus
- Généré des fichiers inverses afin d'indexer les données. Ces fichiers sont directement transposables en base de données (SQL) qui pourront ensuite être interrogées avec de requêtes pour répondre aux questions des utilisateurs.

Nous avons constaté que l'étape de lemmatisation était la plus critique. En effet l'algorithme ne renvoie pas toujours les résultats espérés.

Enfin, l'étape de détermination des mots non significatifs aurait également pu être améliorée.