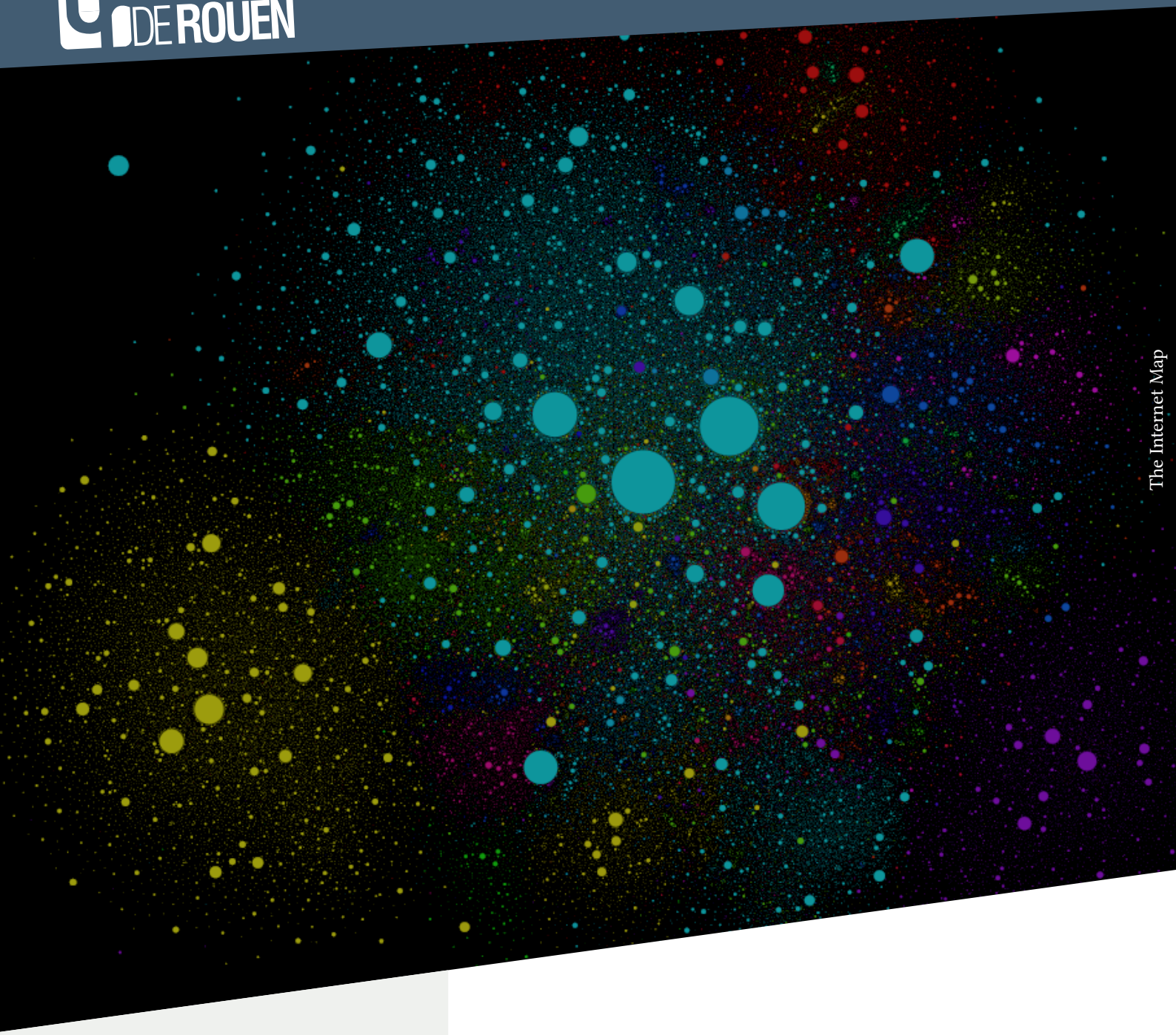
UNIVERSITÉ DE ROUEN



The Internet Map

Année 2014-2015

Thomas Robert

Compte-rendu de TP

Recherche d'Information

Table des matières

1 | Scala

Ce TP/mini-projet a été réalisé en Scala, qui est un langage fonctionnel orienté objet (alors que Java est impératif orienté objet). Scala est un langage compilé en bytecode et exécuté en JVM. Il est donc entièrement compatible avec le code écrit en Java : une librairie Scala est utilisable en Java et une librairie Java est utilisable en Scala.

Les deux avantages principaux de ce langage par rapport à Java dans le cadre de ce TP sont la possibilité d'utiliser les opérations fonctionnelles classiques sur les collections et la légèreté de la syntaxe.

En effet, Scala est fourni avec des collections (listes, ensembles, dictionnaires, etc.) qui possède un ensemble très riche d'opérateurs classiques en programmation fonctionnelle (*map*, *reduce*, *fold*, *filter*, *groupBy*, etc.), qui permettent d'appliquer des transformations sur ces collections avec une syntaxe très légère.

Par exemple, la ligne ci-dessous supprime du dictionnaire urlAndHTMLDatabaseCandidates (associant url au code source de la page) toutes les pages qui ne sont pas du HTML, puis nettoie le code HTML pour en extraire le texte.

```
val urlAndTextDatabase = urlAndHTMLDatabaseCandidates filter {case (_, html) ⇒ isHTML(html)} mapValues cleanHTML
```

Le code Java correspondant serait :

```
Map<String, String> urlAndTextDatabase = new HashMap<String, String>();

for(Entry<String, String> pair: urlAndTextDatabase.entrySet()) {
    String key = pair.getKey();
    String value = pair.getValue();

    if (isHTML(value)) { // filter
        value = cleanHTML(value); // mapValues
        urlAndTextDatabase.put(key, value);
    }
}
```

Soit une syntaxe bien plus lourde, et selon moi bien moins lisible.

L'emploi d'opérateurs fonctionnels courants et de fonctions anonymes permettent donc de transformer des collections de façon très concise.

Scala favorise d'ailleurs globalement la concision, par son mécanisme d'inférence de type qui permet de ne pas avoir à répéter inutilement le type des variables, qui sont déduit de ce que l'on met dans la variable, la légèreté de déclaration des classes, la non-nécessité des points-virgules, etc. [1] Cette concision et puissance du langage permet de passer plus de temps a réfléchir au problème plutôt qu'à taper du code verbeux.

Ce sont donc ces deux avantages majeurs (opérateurs fonctionnels et concision) de Scala qui m'ont le plus servi et « aidé » durant le TP. Mais Scala est un langage complexe et très riche au fonctionnalités bien plus étendues que ce que je viens de présenter.

1. Voir http://www.wikiwand.com/en/Scala_(programming_language)#/Syntactic_flexibility

2 | Sauvegarde des résultats et PageLoader

L'objectif du TP porte sur l'indexation de pages web afin d'y effectuer des recherches. Il est donc nécessaire de télécharger le code source d'un grand nombre de pages web.

Afin de ne pas avoir à recharger plusieurs fois une page web, le module *PageLoader* a été codé. Ce module est appelé pour charger une page, et s'occupe de mettre en cache chaque page chargée. Ainsi, si une page a déjà été chargée (lors d'un lancement précédent), elle sera chargée depuis le cache, permettant de gagner énormément de temps.

Notons que, de manière générale, chaque module sauvegarde son résultat dans un fichier afin que le module suivant puisse être lancé sans avoir besoin de réexécuter les calculs des étapes suivantes.

3 | Crawling

La première étape consiste à crawler le web (suivre les liens de page en page) afin de constituer une « base de données » d'URL.

Ce module de crawling comprend globalement deux parties, une partie qui permet d'extraire et de nettoyer des URL contenues dans une page web, et une partie qui s'occupe de crawler de façon récursive les pages web trouvées.

La première partie n'est pas vraiment intéressante, il s'agit simplement d'être capable reconstruire une URL à partir d'un lien et d'une URL parente, et d'extraire les liens (donc les balises <a>) d'une page web. Tout cela se fait avec des expressions régulières.

La deuxième partie est relativement simple mais plus intéressante.

```
1  def crawl(source: String, limit: Int): Set[String] = {
2
3    def crawlRec(set: Set[String], nextHrefs: List[String]): Set[String] =
       {
4      if (set.size > limit)
5        set
6      else {
7        // crawl new links to add to set
8        val toAdd = getUrls(nextHrefs.head) filter (x ⇒ !(set contains x))
9        // recursive call with new links minus processed link
10       crawlRec(set ++ toAdd, nextHrefs.tail ::: toAdd)
11     }
12   }
13
14   val init = getUrls(source)
15   crawlRec(init.toSet, init)
16 }
```

La fonction de crawling récursive prend une base d'URL et une liste d'URL à parcourir. Elle prend le premier lien de la liste, en extrait les URL enfants, filtre les URL qui ne sont pas déjà dans la base, et ajoute ces URL à la base et à la liste, donc on aura extrait le premier élément puisqu'il a été traité.

On remarque qu'encore une fois, Scala nous permet d'écrire cela très « joliment ».

4 | Indexation

La phase d'indexation consiste, à partir de la base de données d'URL générée par la phase de crawling, à générer un modèle vectoriel tf-idf du corpus.

4.1 Récupération du texte des pages

Pour cela, on commence par récupérer le code source de la page pointée par chaque URL et on vérifie que ce code est du HTML. Si oui, on conserve la page dans le corpus et on extrait de la page HTML le texte brut en utilisant la librairie Jsoup. On dispose donc d'une collection Map[url : String, texte : String].

4.2 Calcul des fréquences des termes dans chaque document *(term frequencies)*

Pour chaque page, on calcule alors les *term frequencies*.

On commence donc par découper les textes à chaque espace afin de récupérer un liste de mots (on ne conserve que les mots constitués de lettres et d'au moins 3 caractères). On *stemme* alors chaque mot de la liste en utilisant un *Stemmer de Porter* en Scala [2].

A partir de la liste de mots *stemmés*, on calcule une *map* du nombre d'occurrence de chaque mot Map[mot : String, nbOccs : Int], et ce pour chaque document.

On traite ensuite chaque Map[mot, nbOccs] pour la transformer en une *map* des poids *tf* en calculant pour chaque nombre d'occurrence log(nbOccs) que l'on divise par le nombre de mots dans le texte.

Au final, on a donc une Map[url : String, Map[mot : String, tf : Double]].

4.3 Calcul des fréquences des termes dans les documents *(inverse document frequencies)*

On veut maintenant calculer le nombre de documents où chaque terme apparait, quel que soit le nombre d'apparitions de ce terme dans chaque document.

Pour cela, on part de notre base de *tf*, Map[url : String, Map[mot : String, tf : Double]]. Pour chaque URL, on extrait la liste des mots contenus dans le document (les clés de la Map[mot, tf]). On concatène toutes les listes obtenues et on a donc une liste dans laquelle un mot apparait à chaque fois qu'il est contenu dans un document.

Il suffit donc de compter le nombre d'occurrences de chaque mot dans cette liste pour savoir dans combien de documents ce mot est présent. On a donc une Map[mot : String, nbOccsDocs : Int].

Enfin, pour calculer les *inverse document frequencies*, on calcule juste pour chaque mot log(nbDocs / nbOccsDocs). On obtient donc Map[mot : String, idf : Double].

2. `https://github.com/aztek/porterstemmer`

4.4 Calcul des vecteurs tf-idf

Enfin, une fonction permet de calculer le vecteur tf-idf normalisé d'un document en multipliant chaque valeur tf par la valeur idf correspondante dans le modèle et en divisant chaque élément par la norme du vecteur.

4.5 Conclusion

A la fin de cette étape, on dispose donc de deux collections, une collection de *term frequencies* Map[url : String, Map[mot : String, tf : Double]] et une collection de *inverse document frequencies* Map[mot : String, idf : Double].

Ces deux collections sont sauvegardées dans des fichiers JSON.

Notons encore une fois que les collections de Scala et son paradigme fonctionnel permettent d'écrire ces transformations de collection de manière très succincte. Par exemple, voici les fonctions permettant de calculer les TF et IDF d'une base d'URL :

```
1  def getNbOccs(wordsList: Iterable[String]): Map[String, Int] = {
2    wordsList groupBy (x ⇒ x) mapValues (_.size)
3  }
4
5  def getFrequencies(wordsList: Iterable[String]): TF = {
6    getNbOccs(wordsList) mapValues (tf ⇒ if (tf == 0) 0 else 1 + Math.log10
       (tf.toDouble) / wordsList.size)
7  }
8
9  def getPageFrequencies(page: String): TF = {
10   getFrequencies(getStems(getWords(page)))
11 }
12
13 def computeTfAndIdf(urlDatabase: Iterable[String]): (TFs, IDF) = {
14
15   // filter to only keep valid HTMLs
16   val urlAndHTMLDatabaseCandidates = (urlDatabase map (url ⇒ url →
       PageLoader.getContent(url)) ).toMap
17   val urlAndTextDatabase = urlAndHTMLDatabaseCandidates filter {case (_,
       html) ⇒ isHTML(html)} mapValues cleanHTML
18   val nbDocs = urlAndTextDatabase.size
19
20   // compute termsFrequencies Map[url, Map[term, tf]]
21   val termsFrequencies = urlAndTextDatabase mapValues getPageFrequencies
22
23   // compute documentsFrequencies Map[term, nbDocsWithTerm]
24   val documentsFrequencies = getNbOccs(termsFrequencies.values flatMap (_
       .keys))
25
26   // compute inverseDocumentsFrequencies Map[term, idf]
27   val inverseDocumentsFrequencies = documentsFrequencies mapValues (
       nbDocsWithTerm ⇒ Math.log10(nbDocs.toDouble / nbDocsWithTerm ))
28
29   (termsFrequencies, inverseDocumentsFrequencies)
30 }
```

5 | Recherche

Enfin, on termine par le module de recherche. Pour cela, on demande à l'utilisateur d'entrer une requête. On calcule le vecteur tf-idf de la requête avec les fonctions du module d'indexation vu précédemment.

On dispose donc des tf-idf du corpus et de la requête, il suffit donc de les comparer par produit vectoriel.

On obtient alors une mesure de similarité requête-document pour chaque document sous la forme d'une Map[url : String, similarité : Double]. Il suffit alors de transformer cette *map* en séquence de paires (url, similarité) que l'on ordonne par paire.

Voici le code permettant de réaliser tout ça :

```
1  def computeSimilarities(queryTfIdf: Indexer.TFIDF, docsTfIdfs: Indexer.
       TFIDFs): Map[String, Double] = {
2    docsTfIdfs mapValues {
3      // ∀ doc, compute cos(q, d) = <q, d> = Σ qi * di
4      dis ⇒ queryTfIdf map {
5        case (qiTerm, qi) ⇒ qi * dis.getOrElse(qiTerm, 0.0d)
6      } reduce (_ + _)
7    }
8  }
9
10 def orderResults(similarities: Map[String, Double]): Seq[(String, Double)
       ] = {
11   similarities.toSeq.sortBy(- _._2)
12 }
13
14 def computeQuery(query: String): Seq[(String, Double)] = {
15   val queryTf = Indexer.getPageFrequencies(query)
16   val queryTfIdf = Indexer.computeTfIdf(queryTf, idf)
17
18   orderResults(computeSimilarities(queryTfIdf, docsTfIdfs))
19 }
```

6 | Résultats

On obtient par exemple le résultat suivant pour la recherche « information » :

```
1  http://research.microsoft.com/en-us/projects/snssurvey/, 0.01955816809945687
2  http://www.thefreelibrary.com/Hohhof%2c+Bonnie-a1, 0.019531222142308673
3  http://www.thefreelibrary.com/Bilal%2c+Dania-a1, 0.01763201349176205
4  https://zh.wikipedia.org/wiki/%E4%BF%A1%E6%81%AF%E6%AA%A2%E7%B4%A2, 0.01629096794038907
5  http://trec.nist.gov, 0.014911142180124952
6  https://fa.wikipedia.org/wiki/%D8%A8%D8%A7%D8%B2%DB%8C%D8%A7%D8%A8%DB%8C_%D8%A7%D8%B7%D9%84%D8%A7%D8%B9%D8%A7%D8%AA, 0.014443995373372875
7  https://tg.wikipedia.org/wiki/%D2%B6%D1%83%D1%81%D1%82%D1%83%D2%B7%D3%AF%D0%B8_%D0%B8%D0%BD%D1%84%D0%BE%D1%80%D0%BC%D0%B0%D1%82%D1%81%D0%B8%D0%BE%D0%BD%D3%A3, 0.013222401253283245
8  http://www.oxforddictionaries.com/social/cite.html?dictCode=english&entryId=information-retrieval, 0.013209528647371774
9  https://el.wikipedia.org/wiki/%CE%91%CE%BD%CE%AC%CE%BA%CF%84%CE%B7%CF%83%CE%B7_%CF%80%CE%BB%CE%B7%CF%81%CE%BF%CF%86%CE%BF%CF%81%CE%B9%CF%8E%CE%BD, 0.013085771367958876
10 https://bg.wikipedia.org/wiki/%D0%98%D0%B7%D0%B2%D0%BB%D0%B8%D1%87%D0%B0%D0%BD%D0%B5_%D0%BD%D0%B0_%D0%B8%D0%BD%D1%84%D0%BE%D1%80%D0%BC%D0%B0%D1%86%D0%B8%D1%8F, 0.012967414816850724
11 http://www.thefreedictionary.com/advertise-with-us.htm, 0.012875194275307 54
12 http://legal-dictionary.thefreedictionary.com/information+resources+management, 0.012680745177095158
13 http://idioms.thefreedictionary.com/information+resources+management, 0.012591283318025044
14 http://legal-dictionary.thefreedictionary.com/information+resources, 0.012495445775518935
15 http://www.britannica.com/topic/1379580/feedback, 0.012114522486594627
16 https://en.wikipedia.org/wiki/Template:Information_science, 0.011964723754558981
17 https://en.wikipedia.org/w/index.php?title=Template:Information_science&amp;action=edit, 0.011964723754558981
18 http://www.britannica.com/print/topic/1379580, 0.011852939474837737
19 http://idioms.thefreedictionary.com/Information+Retrieval, 0.011791158995515177
20 http://financial-dictionary.thefreedictionary.com/Information+retrieval, 0.011783448887460997
21 http://medical-dictionary.thefreedictionary.com/Information+retrieval, 0.011783281512735755
22 http://legal-dictionary.thefreedictionary.com/Information+Retrieval, 0.0117831818092679
23 http://financial-dictionary.thefreedictionary.com/Information+Retrieval, 0.011783154653036983
24 http://medical-dictionary.thefreedictionary.com/Information+Retrieval, 0.011782987907877943
25 https://en.wikipedia.org/wiki/Divergence-from-randomness_model, 0.011754402492824782
```

7 | Conclusion

Ce TP aura donc été l'occasion d'implémenter l'algorithme TF-IDF, sans doute le plus important en recherche d'information et en traitement de documents de manière générale (on peut par exemple l'utiliser en *Machine Learning* par exemple). Cet algorithme a été appliqué à un corpus *crawlé* sur le web.

Cette implémentation pourrait clairement être améliorée. Par exemple, le *stemmer* utilisé est un *stemmer* anglais, mais rien ne nous assurer que les pages crawlées le sont, même si nous sommes partis d'une page anglaise. En l'occurrence, dans les résultats ci-dessus, on voit des pages wikipédia dans de nombreuses langues apparaitre. Il serait par exemple intéressant de détecter la langue de chaque page afin de ne conserver que des pages en anglais.

Il aurait également été intéressant de mesurer les performances en précision et en rappel de notre « moteur de recherche », mais je n'ai pas eu le temps de le faire.

Annexes

A | PageLoader.scala

```scala
package fr.thomasrobert.inforetrieval

import java.nio.file.{Paths, Files}
import java.io.{File, PrintWriter}
import java.net.URL

import scala.io.Source

object PageLoader {

  def getSha1(s: String) = {
    val md = java.security.MessageDigest.getInstance("SHA-1")
    md.digest(s.getBytes("UTF-8")).map("%02x".format(_)).mkString
  }

  def saveAsFile(HTML: String, filename: String) = {
    val writer = new PrintWriter(new File(filename))
    writer.write(HTML)
    writer.close()
  }

  def loadFile(filename: String): String = {
    try {
      Source.fromFile(filename).getLines().mkString("\n")
    }
    catch {
      case e:Exception => ""
    }
  }

  def loadFileAsSet(filename: String): Set[String] = {
    try {
      Source.fromFile(filename).getLines().toSet
    }
    catch {
      case e:Exception => Set()
    }
  }

  def getLines(url: String): Iterator[String] = {
    val connection = new URL(url).openConnection
    connection.setRequestProperty("User-Agent", "Mozilla/4.0 (compatible;
        MSIE 6.0; Windows NT 5.0)")

    Source.fromInputStream(connection.getInputStream).getLines()
  }

  def downloadContent(url: String): String = {
    try
      getLines(url).toList.mkString(" ")
    catch {
      case e:Exception => ""
    }
```

```
53    }
54
55    def getContent(url: String): String = {
56      val filename = "files/"+getSha1(url)+".txt"
57      if (Files.exists(Paths.get(filename)))
58        loadFile(filename)
59      else {
60        val HTML = downloadContent(url)
61        saveAsFile(HTML, filename)
62        HTML
63      }
64    }
65
66  }
```

B | Crawler.scala

```scala
package fr.thomasrobert.inforetrieval

object Crawler {
  /**
   * Return the parts of an URL
   */
  def getInfos(url : String): (String, String, String) = {
    val protocol = ("^([^:]+)://([^/]+)/".r findFirstMatchIn url).get.group(1)
    val domain = ("^([^:]+)://([^/]+)/".r findFirstMatchIn url).get.group(2)
    val directory = ("^(.+/)[^/]*$".r findFirstMatchIn url).get.group(1)
    (protocol, domain, directory)
  }

  /**
   * Get an absolute URL from an HREF and information on the source URL
   */
  def getAbsoluteHref(protocol: String, domain: String, directory: String)(href : String): String = {
    if (matches("[^:]+://.+", href))
      href
    else if (matches("^//.+", href))
      protocol + ":" + href
    else if (matches("^/.+", href))
      protocol + "://" + domain + href
    else
      directory + href
  }

  /**
   * Remove the anchor from an URL
   */
  def removeAnchors(s: String): String = {
    "#.+$".r replaceAllIn(s, "")
  }

  /**
   * Utility function that says if "s" matches the regex string "r"
   */
  def matches(r : String, s : String): Boolean = r.r.pattern.matcher(s).matches()

  /**
   * Extract URLs in the page "parentUrl"
   */
  def getUrls(parentUrl: String): List[String] = {
    try {
      val (protocol, domain, directory) = getInfos(parentUrl)

      val HTML = PageLoader.getContent(parentUrl)
      val HTMLbody = ("<body[^>]+>(.+)</body".r findFirstMatchIn HTML).get.group(1)

      val regex = "href=['\"]([^'\"]+)['\"]".r

      val hrefs = (regex findAllIn HTMLbody) map (x ⇒ (regex
```

```
            findFirstMatchIn x).get.group(1))

      def getAbsoluteHrefLocal = getAbsoluteHref(protocol, domain,
          directory)(_)

      (hrefs map getAbsoluteHrefLocal map removeAnchors).toSet.toList
    }
    catch {
      case e: Exception ⇒ List()
    }
  }

  /**
   * Crawl to look for URLs from a source, stop the crawling when we found
       at least "limit" URLs.
   */
  def crawl(source: String, limit: Int): Set[String] = {

    def crawlRec(set: Set[String], nextHrefs: List[String]): Set[String] =
        {
      if (set.size > limit)
        set
      else {
        // crawl new links to add to set
        val toAdd = getUrls(nextHrefs.head) filter (x ⇒ !(set contains x))
        // recursive call with new links minus processed link
        crawlRec(set ++ toAdd, nextHrefs.tail ::: toAdd)
      }
    }

    val init = getUrls(source)
    crawlRec(init.toSet, init)
  }

  /**
   * Main
   */
  def main(args: Array[String]) {
    PageLoader.saveAsFile(crawl("https://duckduckgo.com/html/?q=information
        %20retrieval", 3000).mkString("\n"), "files/database.txt")
    val database = PageLoader.loadFileAsSet("files/database.txt")
    println(database.size)
    println(database)
  }
}
```

C | Stemmer.scala

```scala
package fr.thomasrobert.inforetrieval

/**
 * Scala implementation of Porter's stemming algorithm.
 *
 * See http://snowball.tartarus.org/algorithms/porter/stemmer.html
 * for description of the algorithm itself.
 *
 * @author Evgeny Kotelnikov <evgeny.kotelnikov@gmail.com>
 */
object Stemmer {
  def stem(word: String): String = {
    // Deal with plurals and past participles
    var stem = new Word(word).applyReplaces(
      "sses" → "ss",
      "ies" → "i",
      "ss" → "ss",
      "s" → "")

    if ((stem matchedBy ((~v~) + "ed")) ||
      (stem matchedBy ((~v~) + "ing"))) {

      stem = stem.applyReplaces(~v~)("ed" → "", "ing" → "")

      stem = stem.applyReplaces(
        "at" → "ate",
        "bl" → "ble",
        "iz" → "ize",
        (~d and not(~L or ~S or ~Z)) → singleLetter,
        (m == 1 and ~o) → "e")
    } else {
      stem = stem.applyReplaces(((m > 0) + "eed") → "ee")
    }

    stem = stem.applyReplaces(((~v~) + "y") → "i")

    // Remove suffixes
    stem = stem.applyReplaces(m > 0)(
      "ational" → "ate",
      "tional" → "tion",
      "enci" → "ence",
      "anci" → "ance",
      "izer" → "ize",
      "abli" → "able",
      "alli" → "al",
      "entli" → "ent",
      "eli" → "e",
      "ousli" → "ous",
      "ization" → "ize",
      "ation" → "ate",
      "ator" → "ate",
      "alism" → "al",
      "iveness" → "ive",
      "fulness" → "ful",
      "ousness" → "ous",
      "aliti" → "al",
      "iviti" → "ive",
```

```
58        "biliti" → "ble")
59
60      stem = stem.applyReplaces(m > 0)(
61        "icate" → "ic",
62        "ative" → "",
63        "alize" → "al",
64        "iciti" → "ic",
65        "ical" → "ic",
66        "ful" → "",
67        "ness" → "")
68
69      stem = stem.applyReplaces(m > 1)(
70        "al" → "",
71        "ance" → "",
72        "ence" → "",
73        "er" → "",
74        "ic" → "",
75        "able" → "",
76        "ible" → "",
77        "ant" → "",
78        "ement" → "",
79        "ment" → "",
80        "ent" → "",
81        ((~S or ~T) + "ion") → "",
82        "ou" → "",
83        "ism" → "",
84        "ate" → "",
85        "iti" → "",
86        "ous" → "",
87        "ive" → "",
88        "ize" → "")
89
90      // Tide up a little bit
91      stem = stem applyReplaces(((m > 1) + "e") → "",
92        (((m == 1) and not(~o)) + "e") → "")
93
94      stem = stem applyReplaces ((m > 1 and ~d and ~L) → singleLetter)
95
96      stem.toString
97    }
98
99    /**
100    * Pattern that is matched against the word.
101    * Usually, the end of the word is compared to suffix,
102    * and the beginning is checked to satisfy a condition.
103    * @param condition Condition to be checked
104    * @param suffix Expected suffix of the word
105    */
106   private case class Pattern(condition: Condition, suffix: String)
107
108   /**
109    * Condition, that is checked against the beginning of the word
110    * @param predicate Predicate to be applied to the word
111    */
112   private case class Condition(predicate: Word ⇒ Boolean) {
113     def + = new Pattern(this, _: String)
114
115     def unary_~ = this // just syntactic sugar
116
117     def ~ = this
```

```
118
119     def and(condition: Condition) = Condition((word) ⇒ predicate(word) &&
            condition.predicate(word))
120
121     def or(condition: Condition) = Condition((word) ⇒ predicate(word) ||
            condition.predicate(word))
122   }
123
124   private def not: Condition ⇒ Condition = {
125     case Condition(predicate) ⇒ Condition(!predicate(_))
126   }
127
128   private val emptyCondition = Condition(_ ⇒ true)
129
130   private object m {
131     def >(measure: Int) = Condition(_.measure > measure)
132
133     def ==(measure: Int) = Condition(_.measure == measure)
134   }
135
136   private val S = Condition(_ endsWith "s")
137   private val Z = Condition(_ endsWith "z")
138   private val L = Condition(_ endsWith "l")
139   private val T = Condition(_ endsWith "t")
140
141   private val d = Condition(_.endsWithCC)
142
143   private val o = Condition(_.endsWithCVC)
144
145   private val v = Condition(_.containsVowels)
146
147   /**
148    * Builder of the stem
149    * @param build Function to be called to build a stem
150    */
151   private case class StemBuilder(build: Word ⇒ Word)
152
153   private def suffixStemBuilder(suffix: String) = StemBuilder(_ + suffix)
154
155   private val singleLetter = StemBuilder(_ trimSuffix 1)
156
157   private class Word(string: String) {
158     val word = string.toLowerCase
159
160     def trimSuffix(suffixLength: Int) = new Word(word substring (0, word.
          length − suffixLength))
161
162     def endsWith = word endsWith _
163
164     def +(suffix: String) = new Word(word + suffix)
165
166     def satisfies = (_: Condition).predicate(this)
167
168     def hasConsonantAt(position: Int): Boolean =
169       (word.indices contains position) && (word(position) match {
170         case 'a' | 'e' | 'i' | 'o' | 'u' ⇒ false
171         case 'y' if hasConsonantAt(position + 1) ⇒ false
172         case _ ⇒ true
173       })
174
```

```
175     def hasVowelAt = !hasConsonantAt(_: Int)
176
177     def containsVowels = word.indices exists hasVowelAt
178
179     def endsWithCC =
180       (word.length > 1) &&
181         (word(word.length - 1) == word(word.length - 2)) &&
182         hasConsonantAt(word.length - 1)
183
184     def endsWithCVC =
185       (word.length > 2) &&
186         hasConsonantAt(word.length - 1) &&
187         hasVowelAt(word.length - 2) &&
188         hasConsonantAt(word.length - 3) &&
189         !(Set('w', 'x', 'y') contains word(word.length - 2))
190
191     /**
192      * Measure of the word -- the number of VCs
193      * @return integer
194      */
195     def measure = word.indices.filter(pos => hasVowelAt(pos) &&
            hasConsonantAt(pos + 1)).length
196
197     def matchedBy: Pattern => Boolean = {
198       case Pattern(condition, suffix) =>
199         endsWith(suffix) && (trimSuffix(suffix.length) satisfies condition)
200     }
201
202     def applyReplaces(replaces: (Pattern, StemBuilder)*): Word = {
203       for ((pattern, stemBuilder) <- replaces if matchedBy(pattern))
204         return stemBuilder build trimSuffix(pattern.suffix.length)
205       this
206     }
207
208     def applyReplaces(commonCondition: Condition)(replaces: (Pattern,
            StemBuilder)*): Word =
209       applyReplaces(replaces map {
210         case (Pattern(condition, suffix), stemBuilder) =>
211           (Pattern(commonCondition and condition, suffix), stemBuilder)
212       }: _*)
213
214     override def toString = word
215   }
216
217   private implicit def pimpMyRule[P <% Pattern, SB <% StemBuilder]
218   (rule: (P, SB)): (Pattern, StemBuilder) = (rule._1, rule._2)
219   private implicit def emptyConditionPattern: String => Pattern = Pattern(
          emptyCondition, _)
220   private implicit def emptySuffixPattern: Condition => Pattern = Pattern(_,
          "")
221   private implicit def suffixedStemBuilder: String => StemBuilder =
          suffixStemBuilder
222 }
```

D | Indexer.scala

```scala
package fr.thomasrobert.inforetrieval

import org.jsoup.Jsoup
import org.json4s.native.Serialization.{read, write}
import org.json4s.DefaultFormats

object Indexer {

  // formats for JSON lib
  implicit val formats = DefaultFormats

  // types
  type IDF    = Map[String, Double]
  type TF     = Map[String, Double]
  type TFIDF  = Map[String, Double]
  type TFs    = Map[String, TF]
  type TFIDFs = Map[String, TFIDF]

  // file save
  def saveAsFile(tfs: TFs, idf: IDF) = {
    PageLoader.saveAsFile(write(tfs), "files/tfs.json")
    PageLoader.saveAsFile(write(idf), "files/idf.json")
  }

  def loadFromFile(): (TFs, IDF) = {
    val tfs = read[TFs](PageLoader.loadFile("files/tfs.json"))
    val idf = read[IDF](PageLoader.loadFile("files/idf.json"))
    (tfs, idf)
  }

  // process HTML & words
  def isHTML(page: String): Boolean = page.toLowerCase.replaceAll("[ \t]",
     "").contains("<body")
  def cleanHTML(page: String): String = Jsoup.parse(page).text

  def getWords(s: String): Iterable[String] = {
    val string = "[^a-zA-Z]".r replaceAllIn (s, " ")
    string split " " filter (x ⇒ x.length > 3) filter (x ⇒ x forall (l ⇒ l.
       isLetter))
  }

  def getStems(words: Iterable[String]): Iterable[String] = {
    words map Stemmer.stem
  }

  // process frequencies (tf & idf)
  def getNbOccs(wordsList: Iterable[String]): Map[String, Int] = {
    wordsList groupBy (x ⇒ x) mapValues (_.size)
  }

  def getFrequencies(wordsList: Iterable[String]): TF = {
    getNbOccs(wordsList) mapValues (tf ⇒ if (tf == 0) 0 else 1 + Math.log10
       (tf.toDouble) / wordsList.size)
  }

  def getPageFrequencies(page: String): TF = {
    getFrequencies(getStems(getWords(page)))
```

```
55    }
56
57    def computeTfAndIdf(urlDatabase: Iterable[String]): (TFs, IDF) = {
58
59      // filter to only keep valid HTMLs
60      val urlAndHTMLDatabaseCandidates = (urlDatabase map (url ⇒ url →
          PageLoader.getContent(url)) ).toMap
61      val urlAndTextDatabase = urlAndHTMLDatabaseCandidates filter {case (_,
          html) ⇒ isHTML(html)} mapValues cleanHTML
62      val nbDocs = urlAndTextDatabase.size
63
64      // compute termsFrequencies Map[url, Map[term, tf]]
65      val termsFrequencies = urlAndTextDatabase mapValues getPageFrequencies
66
67      // compute documentsFrequencies Map[term, nbDocsWithTerm]
68      val documentsFrequencies = getNbOccs(termsFrequencies.values flatMap (_
          .keys))
69
70      // compute inverseDocumentsFrequencies Map[term, idf]
71      val inverseDocumentsFrequencies = documentsFrequencies mapValues (
          nbDocsWithTerm ⇒ Math.log10(nbDocs.toDouble / nbDocsWithTerm ))
72
73      (termsFrequencies, inverseDocumentsFrequencies)
74    }
75
76    def computeTfIdf(tfs: TF, idf: IDF):  TFIDF = {
77      val tfIdfs = tfs map {case (term, tf) ⇒ (term, tf * idf.getOrElse(term,
           0.0d)) }
78      val norm = Math.sqrt( tfIdfs.values.map(x ⇒ x * x).fold(0.0d)(_ + _) )
79      tfIdfs mapValues (_ / norm)
80    }
81
82    def computeTfIdfs(tfs: TFs, idf: IDF):  TFIDFs = {
83      tfs mapValues (tf ⇒ computeTfIdf(tf, idf))
84    }
85
86    // main
87    def main(args: Array[String]) {
88
89      val (termsFrequencies, inverseDocumentsFrequencies) = computeTfAndIdf(
          PageLoader.loadFileAsSet("files/database.txt"))
90      saveAsFile(termsFrequencies, inverseDocumentsFrequencies)
91      //  val (termsFrequencies, inverseDocumentsFrequencies) = loadFromFile
          ()
92
93      println(termsFrequencies.size+" docs")
94      println(termsFrequencies.head)
95      println(inverseDocumentsFrequencies)
96    }
97  }
```

E | Search.scala

```scala
package fr.thomasrobert.inforetrieval

object Search {

  val (tfs, idf) = Indexer.loadFromFile()
  val docsTfIdfs = Indexer.computeTfIdfs(tfs, idf)

  def computeSimilarities(queryTfIdf: Indexer.TFIDF, docsTfIdfs: Indexer.TFIDFs): Map[String, Double] = {
    docsTfIdfs mapValues {
      // ∀ doc, compute cos(q, d) = <q, d> = Σ qi * di
      dis ⇒ queryTfIdf map {
        case (qiTerm, qi) ⇒ qi * dis.getOrElse(qiTerm, 0.0d)
      } reduce (_ + _)
    }
  }

  def orderResults(similarities: Map[String, Double]): Seq[(String, Double)] = {
    similarities.toSeq.sortBy(- _._2)
  }

  def computeQuery(query: String): Seq[(String, Double)] = {
    val queryTf = Indexer.getPageFrequencies(query)
    val queryTfIdf = Indexer.computeTfIdf(queryTf, idf)

    orderResults(computeSimilarities(queryTfIdf, docsTfIdfs))
  }

  // main
  def main(args: Array[String]): Unit = {
    var continue = true
    while (continue) {
      val ln = scala.io.StdIn.readLine()
      if (ln != null)
        computeQuery(ln) take 25 map println
      else
        continue = false
    }
  }

}
```