Pràctica Scala
Informe

Marc Cané i Enric Rodríguez

03/12/2018

Índex

Capítol 1

Principals funcions d'ordre superior usades

Recopilatori amb exemples de les principals funcions d'ordre superior usades a l'hora de realitzar aquesta pràctica.

Map

$map :: (a \to b) \to [a] \to [b]$

La funció d'ordre superior més utilitzada és sense cap mena de dubte la funció *map*. Tot i que principalment s'usa per convertir les dades, en algun cas també la usem per realitzar operacions aritmètiques sobre dues llistes representant vectors.

La primera aplicació de la funció que se'ns va passar pel cap va ser la de filtrar els fitxers d'entrada.

```
/* Given a filename, produces a representative string that only contains lower case
    characters and spaces
*  @param filename the file path of the file to be readed
*
*  @pre File must exist, otherwise will throw an exception
*/
def readFile(filename : String) : String = {
   val source = scala.io.Source.fromFile(filename, "UTF-8")
   val str = try source.map(c => if(acceptableChar(c)) c else ' ').mkString finally
       source.close()
   str.toLowerCase.trim
}
```

```
val str = try contingut.map(c => if(FirstHalf.acceptableChar(c)) c else ' ').mkString
    finally xmlleg.close()
```

Aquesta funció la usem molt durant el comput de la similitud cosinus entre dos documents. En aquest cas la usem, en ordre descendent, per:

- A *euclidean_ norm*, per elevar al quadrat tota una col·lecció de nombres
- A *cosinesim*, per a normalitzar els vectors
- A *cosinesim*, per a deixar a 0 un vector (usat per a alinear-los)
- A *cosinesim*, per a simplificar els vectors de tuples a vectors de nombres
- A *cosinesim*, per multiplicar les tuples de nombres generades per la funció *zip*

```
/* Computes the euclidean norm of a vector (of Double values)
 * @param v the Vector
 */
def euclidean_norm(v: Iterable[Double]) = {
  Math.sqrt( v.map( x => x*x ).reduceLeft( _ + _ ) )
}

/* Given two strings representing two filtered documents, and a list of words that
    will be filtered, it computes its cosine similarity
 *  @param s1 First document
 *  @param s2 Second document
 *  @param stopwords List of words that will be discarded
 */
def cosinesim(s1: String, s2: String, stopwords: List[String]): Double = {
  val freq1 = nonstopfreq(s1, stopwords).toMap;
  val freq2 = nonstopfreq(s2, stopwords).toMap;
  val freq1max = freq1.values.max
  val freq2max = freq2.values.max

  //normalizing vectors
  val freq1norm = freq1.map{ case (key, value) => (key, value.toDouble/freq1max) }
  val freq2norm = freq2.map{ case (key, value) => (key, value.toDouble/freq2max) }

  //aligning vectors
  val smv1 = freq2norm.map({ case (key, value) => (key, 0.0)}) ++ freq1norm
  val smv2 = freq1norm.map({ case (key, value) => (key, 0.0)}) ++ freq2norm

  //simplify vectors
  val smv1_vec = smv1.values.map(x => x.asInstanceOf[Double])
  val smv2_vec = smv2.values.map(x => x.asInstanceOf[Double])

  //compute cosinesim
  val term = smv1_vec.zip(smv2_vec).map(x => x._1 * x._2).reduceLeft( _ + _ )

  term / (euclidean_norm(smv1_vec) * euclidean_norm(smv2_vec))
}
```

En aquest exemple també es veu una de les altres funcions més usades: *reduce*

També la fem servir quan volem realitzar les mateixes operacions a la segona part amb el patró MapReduce. S'usa múltiples cops a la funció reducing del primer dels MapReduce que fem per calcular el tf_idf.

```
//key-> Filename, values-> list of (Word, count)
/*   Given a file name and a list of tuples (Word, 1) (word occurrences) of that file,
    reduces the tuples with the same first value to a tuple (Word, n_ocurrences)
 *   @param key Name of the file
 *   @param values List of all occurrences of the words
 */
def reducing1(key: String, values: List[(String, Double)]): List[(String, Double)] = {
  val res = for( (word, count_list) <- values.groupBy(_._1).toList )
    //For every pair of word and list of counts, add up its counts
    yield (word, count_list.map( {case (_, count) => count } ).reduceLeft( _ + _))

  val max_freq = res.map(_._2).max

  res.sortWith(FirstHalf.moreFrequent).map(x => (x._1, x._2/max_freq))
}
```

O quan volem fer l'index invers per contar a quants fitxers surt una paraula determinada. Aquí també usem la funció *map* (concretament a la funció de mapping del segon MapReduce).

```
/* Given a file name and its word counts, unfolds all its tuples to the reverse (Word,
    File) tuple.
 *   This will let us count how many documents contain that word and compute the idf of
     that word.
 *   @param file_name Name of that file
 *   @param word_count List of word counts (Word, 1)
 */
def mapping2(file_name: String, word_count: List[(String, Double)]): List[(String,
    String)] = {
  word_count.map(x => (x._1, file_name))
}
```

Tenim una adaptació de la funció *cosinesim* per a usar-la a la segona part amb els MapReduce. Com era d'esperar, aquesta adaptació usa de la mateixa manera que la primera la funció *map*.

```
/* Given two maps representing the frequency counts of the words of two files, it
    computes its cosine similarity
 *    @param m1 First document
 *    @param m2 Second document
 */
def cosinesim2(m1: Map[String, Double], m2: Map[String, Double]): Double = {

  //aligning vectors
  val smv1 = m2.map({ case (key, value) => (key, 0.0)}) ++ m1
  val smv2 = m1.map({ case (key, value) => (key, 0.0)}) ++ m2

  //simplify vectors
  val smv1_vec = smv1.values.map(x => x.asInstanceOf[Double])
  val smv2_vec = smv2.values.map(x => x.asInstanceOf[Double])

  //compute cosinesim
  val term = smv1_vec.zip(smv2_vec).map(x => x._1 * x._2).reduceLeft( _ + _ )

  term / (FirstHalf.euclidean_norm(smv1_vec) * FirstHalf.euclidean_norm(smv2_vec))
}
```

A vegades ens interessa fer un petit processat de la input abans de passarla al MapReduce. En aquestes situacions la funció *map* ens és molt útil.

```
//// 3/4 Computing TF_IDF ////

//Computing the idf of every word
//this line could be done with MapReduce, but the overhead caused by map and reduce
    actor initialization is not worth the time
val idf = df.map(x => (x._1, Math.log(tf.size/x._2.length)))

//preparing input: adding the idf to all input values
//input: List[File -> ( List[(Word, dtf)], List[(Word, idf)])]
val tfIdfInput = tf.map({case (k,v) => (k, (v,idf))})
```

El mateix podem dir per el resultat final.

```
//final product: Map[(Filename, Filename), cosinesim]
     val finalResult = result.map({case (k,v) => (k, v.apply(0))})
```

I per acabar, al segon apartat de la segona part de la pràctica usem la funció *map* per eliminar els claudàtors inicials i finals, i posteriorment la usem un parell de cops més per tallar la string i agafar només el títol del document al que fa referencia, eliminant sobrenoms dels documents i noms de secció del document.

```
//The order of the operations is important. There'll be references to pages that we
    don't have
val kk3 = refs.filterNot(x=> x.contains(':') ||
    x.apply(2)=='#').map(x=>x.substring(2,x.length()-2))
    .map(x=>x.split("\\|").apply(0)).map(x=>x.split("#").apply(0))
```

Filter i FilterNot

$filter :: (a \to Boolean) \to [a] \to [a]$
$filterNot :: (a \to Boolean) \to [a] \to [a]$

La funció *filter*, i la seva germana *filterNot*, són funcions que ens permeten filtrar el contingut d'una col·lecció donant una funció de filtratge que nosaltres definim lliurement. El primer cas d'ús d'aquesta al llarg de la nostra pràctica és un bastant esperable: filtrar els fitxers d'entrada per extensió.

```
/* Given a folder and two strings with the prefix and sufix of the files to be opened it
    returns an array of file handlers
   *  @param folder Name of the folder to search the files in
   *  @param startsWith Prefix of the files to be opened
   *  @param endsWith Sufix of the files to be opened
   */
  def openFiles(folder: String, startsWith: String, endsWith: String):
      Array[java.io.File] = {
    var fileList = new java.io.File(folder).listFiles
    .filter(_.getName.startsWith(startsWith))
    .filter(_.getName.endsWith(endsWith))
    fileList
  }
```

El següent ús és per filtrar les stopwords del càlcul de freqüències.

```
/* Given a string @p s and a list of excluded words @stopwords, evaluates as the
    absolute frequencies of the words (substrings spaced by white spaces ' ') that @p s
    contains and @p stopwords does not
   * More specifically, it evaluates as a list of pairs (String, Int), being the String a
       word of @p s but not a word of @p stopwords, and Int being its absolute frequency
   * @param s A string
   * @param stopwords A list of strings
   */
  def nonstopfreq (s: String, stopwords: List[String]) =
    freq(s).filterNot(x => stopwords.contains(x._1))
```

```
/* First mapping function. Given a file name (not the path) and a tuple (File_path,
    List[stopwords])
    *   generates all the pairs (file name, (word, 1)). This will let us reduce the list
        to a list of ocurrences (frequency)
    *   @param file_name The file name
    *   @param file Tuple of (filePath, List[stopwords])
    */
   def mapping1(file_name: String, file: (String, List[String])): List[(String, (String,
       Double))] = {
     val wordList = tractaxmldoc.readXMLFile(file._1).split("
         +").toList.filterNot(file._2.contains(_))

     val x = (for(word <- wordList) yield (file_name, (word, 1.toDouble)))
     x
   }
```

Posteriorment la usem per filtrar els documents per llindars de similitud i per trobar els documents que es referencien i no es referencien entre si.

```
 val Llindar: Double = 0.2

 val stopwords = FirstHalf.readFile("stopwordscat-utf8.txt").split(" +").toList
 val tf_idf = MapReduceTfIdf.computeSimilarities(files, stopwords, 10, 10, false)
 val tf_idf2 = tf_idf.filter(x => x._2 > Llindar)

 val fileMap = tractaxmldoc.titolsNomfitxer(files).toMap
 val tf_idf3 = tf_idf2.filter(x =>
     result2mapfilled(fileMap(x._1._1)).contains(fileMap(x._1._2)) )

 println("10 primeres parelles de pagines similars que no es referencien una a l'altra")
 for(i <- tf_idf3.take(10)) println(i)


 val tf_idf4 = tf_idf.filter(x => x._2 < Llindar)
 val tf_idf5 = tf_idf4.filterNot(x =>
     result2mapfilled(fileMap(x._1._1)).contains(fileMap(x._1._2)) )
```

I per acabar, també la fem servir quan volem treure els enllaços a fitxers multimèdia durant el filtratge a de referencies.

```
//The order of the operations is important. There'll be references to pages that we
    don't have
   val kk3 = refs.filterNot(x=> x.contains(':') ||
       x.apply(2)=='#').map(x=>x.substring(2,x.length()-2))
    .map(x=>x.split("\\|").apply(0)).map(x=>x.split("#").apply(0))
```

Reduce i Fold

$reduceLeft :: (a \to a \to a) \to [a] \to [a]$
$foldLeft :: a \to (a \to a \to a) \to [a] \to [a]$

Si has arribat fins aquí, segurament hauràs vist que *reduceLeft* és una de les funcions que més es repeteixen. I és que també la usem múltiples vegades. També fem servir la funció *foldLeft*, que és molt semblant *reduceLeft* però ens permet triar l'element inicial amb el que començar el "plegat"de la col·lecció. Aquesta última només es fa servir un cop a l'hora de mostrar els resultats de la primera part, i podria ser substituïda per *reduceLeft*.

```
/*
    * Given a list of tuples (_ Int), it prints the tuple in a fancy way and lists the
        division between _._2 and the total sum of _._2 of all the tuples on the list
    * In our case, we use this function to print the list of frequencies of certain
        words in a file, being _._2 the absolute count of occurrences of this word.
    * @param frequencyList List of tuples (_, Int)
    * @param n Number of elements to be shown
    */
   def topN(freqencyList: List[(_, Int)], n: Int) = {
     val nWords = freqencyList.foldLeft(0) { (total, actual) => total + actual._2 }
     val nDiffWords = freqencyList length;
     println("N Words: " + nWords + " Diferent: " + nDiffWords)
     printf("%-30s %-11s %-10s\n", "Words", "ocurrences", "frequency")
     for(r <- freqencyList.slice(0,n)) printf("%-30s %-11d %-10.7f%%\n", r._1, r._2,
         (r._2.toFloat/nWords)*100) //println(r._1 + "   " + r._2 + " " +
         (r._2.toFloat/nWords)*100)
     println()
   }
```

El primer ús de *reduceLeft* el podem trobar quan volem calcular la norma euclidiana d'un vector de nombres. Amb reduce podem sumar fàcilment tots els elements del sumatori.

```
/* Computes the euclidean norm of a vector (of Double values)
*  @param v the Vector
*/
def euclidean_norm(v: Iterable[Double]) = {
   Math.sqrt( v.map( x => x*x ).reduceLeft( _ + _ ) )
}
```

També s'usa per calcular el producte escalar dels dos vectors de freqüències (terme de la formula de la similitud cosinus).

```
   val term = smv1_vec.zip(smv2_vec).map(x => x._1 * x._2).reduceLeft( _ + _ )
```

I per últim, la fem servir a l'hora de computar el nombre d'ocurrències d'una paraula en un document amb MapReduce (primera funció de reducció).

```
//key-> Filename, values-> list of (Word, count)
   /*   Given a file name and a list of tuples (Word, 1) (word occurrences) of that
        file, reduces the tuples with the same first value to a tuple (Word, n_ocurrences)
    *   @param key Name of the file
    *   @param values List of all occurrences of the words
    */
   def reducing1(key: String, values: List[(String, Double)]): List[(String, Double)] = {
     val res = for( (word, count_list) <- values.groupBy(_._1).toList )
       //For every pair of word and list of counts, add up its counts
       yield (word, count_list.map( {case (_, count) => count } ).reduceLeft( _ + _))

     val max_freq = res.map(_._2).max

     res.sortWith(FirstHalf.moreFrequent).map(x => (x._1, x._2/max_freq))
   }
```

SortWith

$sortWith :: (a \to a \to Boolean) \to [a] \to [a]$

La funció *sortWith* ens permet ordenar una col·lecció passant com a paràmetre una funció d'ordenació definida per nosaltres. Principalment la usem per ordenar els resultats de les nostres funcions abans de mostrar-los.

```
/*
    *                   Evaluating freq() function
    */

   Statistics.topN(freq(pg11).sortWith(moreFrequent), 10)

   /*
    *                   Evaluating nonstopfreq() function
    */

   Statistics.topN(nonstopfreq(pg11, english_stopwords).sortWith(moreFrequent), 10)

   /*
    *                   Evaluating paraulesfreqfreq() function
    */

   Statistics.paraulafreqfreqStats(paraulafreqfreq(pg11).sortWith(_._1 < _._1), 10, 5)

    /*
    *                   Evaluating ngrams() function
    */

   Statistics.topN(ngramsfreq(pg11, 3).sortWith(moreFrequent), 10)
```

I també l'usa la funció *reducing1* amb el resultat final del seu comput per ordenar per freqüència.

```
   res.sortWith(FirstHalf.moreFrequent).map(x => (x._1, x._2/max_freq))
```

Capítol 2

Creació dels MapReduce

Computar similaritats

Aquesta implementació del MapReduce per resoldre el problema de trobar la similitud entre documents fa ús de 4 instàncies consecutives d'aquest.

El primer busca els vectors de freqüències absolutes de tots els fitxers que es volen comparar. La funció de mapping crea les parelles (fitxer, (paraula, 1)) i la funció reduce busca acumular totes aquestes tuples i deixar-ho en (fitxer, List[(paraula, ocurrències)]).

```
/* First mapping function. Given a file name (not the path) and a tuple (File_path,
    List[stopwords])
 * generates all the pairs (file name, (word, 1)). This will let us reduce the list
    to a list of ocurrences (frequency)
 * @param file_name The file name
 * @param file Tuple of (filePath, List[stopwords])
 */
def mapping1(file_name: String, file: (String, List[String])): List[(String, (String,
    Double))] = {
  val wordList = tractaxmldoc.readXMLFile(file._1).split("
      +").toList.filterNot(file._2.contains(_))

  val x = (for(word <- wordList) yield (file_name, (word, 1.toDouble)))
  x
}

//key-> Filename, values-> list of (Word, count)
/* Given a file name and a list of tuples (Word, 1) (word occurrences) of that file,
    reduces the tuples with the same first value to a tuple (Word, n_ocurrences)
 * @param key Name of the file
 * @param values List of all occurrences of the words
 */
def reducing1(key: String, values: List[(String, Double)]): List[(String, Double)] = {
  val res = for( (word, count_list) <- values.groupBy(_._1).toList )
    //For every pair of word and list of counts, add up its counts
    yield (word, count_list.map( {case (_, count) => count } ).reduceLeft( _ + _))

  val max_freq = res.map(_._2).max

  res.sortWith(FirstHalf.moreFrequent).map(x => (x._1, x._2/max_freq))
}
```

A partir d'aquí, tot i que s'usa el patró MapReduce, no s'implementen funcions de reduce ja

que de la manera que ho hem plantejat no són necessàries. En comptes d'una funció personalitzada usem una funció genèrica que serveix de passarel·la per a un dels seus paràmetres.

```
/* Doesn't reduce. It's simply a pass through for the @p second parameter
 *    @param first First parameter
 *    @param second Second parameter
 */
def passThroughtReduce[A](first: Any, second: List[A]) = {
  second
}
```

D'aquesta manera podem seguir usant el patró MapReduce, tot i que no és la solució més eficient.

La segona funció de mapping pretén trobar l'index invers per poder calcular l'idf.

```
    /*  Given a file name and its word counts, unfolds all its tuples to the reverse
        (Word, File) tuple.
     *  This will let us count how many documents contain that word and compute the idf
         of that word.
     *  @param file_name Name of that file
     *  @param word_count List of word counts (Word, 1)
     */
    def mapping2(file_name: String, word_count: List[(String, Double)]): List[(String,
        String)] = {
      word_count.map(x => (x._1, file_name))
    }
```

La tercera funció de mapping computa els tf_idf de tots els termes d'un fitxer donat.

```
/*  Given a file name, and a tuple containing the List of word counts and a Map with (at
     least) the idf of all words in that file
     *  yields all the tuples (file name, (Word, tfidf)) for that list of word counts.
         This will let us compute the space model vector
     *  of that file
     *  @param file_name Name of the file
     *  @param tf_idf_unfolded Pair containing the frequency counts of that file and the
         idf of all the words in that document set
     */
    def mapping3(file_name: String, tf_idf_unfolded: (List[(String, Double)], Map[String,
        Double])): List[(String, (String,Double))] = {
      for (term_freq <- tf_idf_unfolded._1)
        yield ( (file_name, (term_freq._1, term_freq._2 *
            tf_idf_unfolded._2(term_freq._1))))
    }
```

I per acabar, l'última funció de mapping computa la similitud cosinus entre dos fitxers donats els seus vectors tf_idf. Usa una funció *cosinesim* adaptada per a fer-la servir de manera més eficient.

```
/*  Given two maps representing the frequency counts of the words of two files, it
    computes its cosine similarity
    *  @param m1 First document
    *  @param m2 Second document
    */
   def cosinesim2(m1: Map[String, Double], m2: Map[String, Double]): Double = {

     //aligning vectors
     val smv1 = m2.map({ case (key, value) => (key, 0.0)}) ++ m1
     val smv2 = m1.map({ case (key, value) => (key, 0.0)}) ++ m2

     //simplify vectors
     val smv1_vec = smv1.values.map(x => x.asInstanceOf[Double])
     val smv2_vec = smv2.values.map(x => x.asInstanceOf[Double])

     //compute cosinesim
     val term = smv1_vec.zip(smv2_vec).map(x => x._1 * x._2).reduceLeft( _ + _ )

     term / (FirstHalf.euclidean_norm(smv1_vec) * FirstHalf.euclidean_norm(smv2_vec))
   }

   /*  Given a pair of file names, and its representing space model vectors, computes
       its cosine similarity
    *  @param files Pair of file names
    *  @param tf_idfs Pair of space model vectors
    */
   def mapping4(files: (String, String), tf_idfs:(List[(String, Double)], List[(String,
       Double)]) ) = {
     val cosinesim = cosinesim2(tf_idfs._1.toMap, tf_idfs._2.toMap)

     List((files, cosinesim))
   }
```

Parells de pagines similars sense referències i viceversa

El primer MapReduce fa les referències inverses de cada document, és a dir, que troba tots els documents que fan referència a ell. En la funció de mapping generem cada parella (document, document que fa referència a l'anterior). En el reduce només descartem els repetits, per el cas de que un document tingués vàries referències a un altre.

```
def mapping1(title: String, referencedDocs: List[String]): List[(String, String)] = {
  for(doc <- referencedDocs) yield (doc, title) //document, document that references
      the first one
}

def reducing1(title: String, docsRefer: List[String]): List[String] = {
  docsRefer.distinct
}
```

A continuació quedava descartar totes les referències de documents que no teníem, eliminar les referències reflexives, crear entrades pels documents que no eren referenciats, cridar la funció per calcular les similituds, filtrar els resultats amb el llindar i que no estiguessin relacionats.

Aquest MapReduce havia d'invertir les referències als documents però no va funcionar com esperàvem i no ens va quedar temps per arreglar-lo.

```
def mapping2(title: String, tuple: (List[String],List[String])): List[(String,
    String)] = {
  val allTitles = tuple._2
  for(titleRef <- allTitles; if(!tuple._1.contains(titleRef))) yield (title, titleRef)
}

def reducing2(title: String, docsRefer: List[String]): List[String] = {
  docsRefer
}
```

Capítol 3

Classes usades

Hem fet servir una classe MapReduceActor que implementa el patró MapReduce usant la llibreria d'Scala AKKA actors. És una classe genèrica sobre 4 tipus i que té una serie de paràmetres. Aquests són els vistos a classe amb el mètode mapreduce. La classe estén la classe Actor, i per tant, també n'és un. Es podrien fer varis MapReduce simultàniament.

```
class MapReduceActor[K, V, K2, V2]
    (
    input:   List[(K, V)],
    mapping: (K, V) => List[(K2, V2)],
    reducing: (K2, List[V2]) => List[V2],
    numMappers: Int,
    numReducers: Int                    ) extends Actor{
```

Aquesta classe conte a dins un conjunt de subclasses necessàries per al correcte funcionament d'aquesta.

```
//master - service messages
case class MapOrder(k: K, v: V, mapping: (K, V) => List[(K2, V2)])
case class ReduceOrder(k: K2, vlist: List[V2], reducing: (K2, List[V2]) => List[V2])

//service - master messages
case class Intermediate(list: List[(K2, V2)])
case class Reduced(key: K2, list: List[V2])
```

Aquestes classes s'usen per al pas de missatges master - service i viceversa. Transporten la input pendent per processar i els resultats, respectivament. A més, hi han instanciacions d'objectes anònims que fan d'actors de mapping i reducing, però no tenen una classe de per si.

Capítol 4

Resultats d'execució

Resultats de la primera part

A la nostra aplicació podrem trobar un objecte anomenat FirstHalf que conté tot el codi corresponent a la primera part de la pràctica. Aquest objecte té un mètode anomenat *main* i conté un codi que prova el funcionament d'aquesta primera part. Per executar-lo hem col·locat tots els documents proporcionats per el professor a una carpeta anomenada *test*. El que fa és executar les mateixes proves que es mostren al enunciat de la primera part, i posteriorment comprova la similitud cosinus entre tots els fitxers de la carpeta test que comencen per pg i tenen extensió .txt.

El codi d'aquesta funció és el següent.

```
def main() = {

    //load stopwords
    val english_stopwords = readFile("test/english-stop.txt").split(" +").toList

    //load pg11 and pg11-net
    val pg11 = readFile("test/pg11.txt")
    val pg11_net = readFile("test/pg11-net.txt")

    //load all the files that will be evaluated with cosinesim()
    val fileSet = new java.io.File("test").listFiles.filter((x) =>
        x.getName.startsWith("pg") && x.getName.endsWith(".txt")).toSet


    /*
     *                   Evaluating freq() function
     */

    Statistics.topN(freq(pg11).sortWith(moreFrequent), 10)

    /*
     *                   Evaluating nonstopfreq() function
     */

    Statistics.topN(nonstopfreq(pg11, english_stopwords).sortWith(moreFrequent), 10)

    /*
     *                   Evaluating paraulesfreqfreq() function
     */

    Statistics.paraulafreqfreqStats(paraulafreqfreq(pg11).sortWith(_._1 < _._1), 10, 5)

     /*
     *                   Evaluating ngrams() function
     */

    Statistics.topN(ngramsfreq(pg11, 3).sortWith(moreFrequent), 10)
```

```
  /*
   *                    Evaluating cosinesim() function
   */

  println(s"Similitud cosinus entre totes les permutacions dels fitxers:
      \n''${fileSet.mkString("'',''")}''")
  println("| ")

  val t0 = System.nanoTime

  for(pair <- fileSet.subsets(2)){
    val t1 = System.nanoTime
    val file_1 = pair.head
    val file_2 = pair.tail.head
    val file_1_str = readFile(file_1.getAbsolutePath)
    val file_2_str = readFile(file_2.getAbsolutePath)
    print("| ")
    Statistics.cosineSimStat(file_1.getName, file_2.getName, cosinesim(file_1_str,
        file_2_str, english_stopwords))
    println("| Calculada en temps: " + (System.nanoTime-t1)/Math.pow(10,9) + " segons")
  }

  println("Temps total per calcular totes les comparacions: " +
      (System.nanoTime-t0)/Math.pow(10,9) + " segons")

}
```

I ens genera la següent sortida.

```
N Words: 30419 Diferent: 3007
Words                          ocurrences  frequency
the                            1818        5,9765282 %
and                            940         3,0901740 %
to                             809         2,6595221 %
a                              690         2,2683191 %
of                             631         2,0743613 %
it                             610         2,0053256 %
she                            553         1,8179427 %
i                              545         1,7916434 %
you                            481         1,5812486 %
said                           462         1,5187876 %

N Words: 10038 Diferent: 2623
Words                          ocurrences  frequency
alice                          403         4,0147438 %
gutenberg                      93          0,9264794 %
project                        87          0,8667065 %
queen                          75          0,7471608 %
thought                        74          0,7371987 %
time                           71          0,7073122 %
king                           63          0,6276151 %
turtle                         59          0,5877665 %
began                          58          0,5778043 %
tm                             57          0,5678422 %

Les 10 frequencies mes frequents:
1330 paraules apareixen 1
468 paraules apareixen 2
264 paraules apareixen 3
176 paraules apareixen 4
101 paraules apareixen 5
72 paraules apareixen 6
66 paraules apareixen 7
74 paraules apareixen 8
39 paraules apareixen 9
35 paraules apareixen 10
Les 5 frequencies menys frequents:
1 paraules apareixen 631 vegades
1 paraules apareixen 690 vegades
1 paraules apareixen 809 vegades
1 paraules apareixen 940 vegades
1 paraules apareixen 1818 vegades
```

```
N Words: 30417 Diferent: 25774
Words                          ocurrences  frequency
project gutenberg tm           57          0,1873952 %
the mock turtle                53          0,1742447 %
i don t                        31          0,1019167 %
the march hare                 30          0,0986290 %
said the king                  29          0,0953414 %
the project gutenberg          29          0,0953414 %
said the hatter                21          0,0690403 %
the white rabbit               21          0,0690403 %
said the mock                  19          0,0624651 %
said to herself                19          0,0624651 %
```

```
Similitud cosinus entre totes les permutacions dels fitxers:
''test\pg74.txt'',''test\pg74-net.txt'',''test\pg11.txt'',''test\pg12-net.txt'',''test\pg2500.txt'',
|
| La similitud cosinus entre pg74.txt i pg74-net.txt es de 0,9889848979
| Calculada en temps: 1.0292793 segons
| La similitud cosinus entre pg74.txt i pg11.txt es de 0,2586942062
| Calculada en temps: 0.5597866 segons
| La similitud cosinus entre pg74.txt i pg12-net.txt es de 0,2084141438
| Calculada en temps: 0.4712166 segons
| La similitud cosinus entre pg74.txt i pg2500.txt es de 0,2993298356
| Calculada en temps: 0.516914 segons
| La similitud cosinus entre pg74.txt i pg11-net.txt es de 0,2178060109
| Calculada en temps: 0.3915398 segons
| La similitud cosinus entre pg74.txt i pg12.txt es de 0,2472302429
| Calculada en temps: 0.4011614 segons
| La similitud cosinus entre pg74.txt i pg2500-net.txt es de 0,2679095018
| Calculada en temps: 0.4061952 segons
| La similitud cosinus entre pg74-net.txt i pg11.txt es de 0,2149599531
| Calculada en temps: 0.3756475 segons
| La similitud cosinus entre pg74-net.txt i pg12-net.txt es de 0,2097871125
| Calculada en temps: 0.3173072 segons
| La similitud cosinus entre pg74-net.txt i pg2500.txt es de 0,2654043911
| Calculada en temps: 0.3413635 segons
| La similitud cosinus entre pg74-net.txt i pg11-net.txt es de 0,2193384017
| Calculada en temps: 0.3003146 segons
| La similitud cosinus entre pg74-net.txt i pg12.txt es de 0,2087121450
| Calculada en temps: 0.3352347 segons
| La similitud cosinus entre pg74-net.txt i pg2500-net.txt es de 0,2689909063
| Calculada en temps: 0.4876476 segons
| La similitud cosinus entre pg11.txt i pg12-net.txt es de 0,8238081983
| Calculada en temps: 0.2270051 segons
| La similitud cosinus entre pg11.txt i pg2500.txt es de 0,2768360416
| Calculada en temps: 0.213087 segons
| La similitud cosinus entre pg11.txt i pg11-net.txt es de 0,9516910223
| Calculada en temps: 0.1998175 segons
| La similitud cosinus entre pg11.txt i pg12.txt es de 0,8764098616
| Calculada en temps: 0.1851824 segons
| La similitud cosinus entre pg11.txt i pg2500-net.txt es de 0,2083246560
| Calculada en temps: 0.2210432 segons
```

```
| La similitud cosinus entre pg12-net.txt i pg2500.txt es de 0,1984522576
| Calculada en temps: 0.2212506 segons
| La similitud cosinus entre pg12-net.txt i pg11-net.txt es de 0,8641992087
| Calculada en temps: 0.169078 segons
| La similitud cosinus entre pg12-net.txt i pg12.txt es de 0,9625994241
| Calculada en temps: 0.2217224 segons
| La similitud cosinus entre pg12-net.txt i pg2500-net.txt es de 0,2022974710
| Calculada en temps: 0.1981073 segons
| La similitud cosinus entre pg2500.txt i pg11-net.txt es de 0,2088845337
| Calculada en temps: 0.196527 segons
| La similitud cosinus entre pg2500.txt i pg12.txt es de 0,2611308549
| Calculada en temps: 0.2370576 segons
| La similitud cosinus entre pg2500.txt i pg2500-net.txt es de 0,9713523359
| Calculada en temps: 0.2717254 segons
| La similitud cosinus entre pg11-net.txt i pg12.txt es de 0,8322888009
| Calculada en temps: 0.2073292 segons
| La similitud cosinus entre pg11-net.txt i pg2500-net.txt es de 0,2131869392
| Calculada en temps: 0.2549333 segons
| La similitud cosinus entre pg12.txt i pg2500-net.txt es de 0,2008062998
| Calculada en temps: 0.2815157 segons
Temps total per calcular totes les comparacions: 9.2560514 segons
```

Resultats de la segona part

Resultats del càlcul de similitud entre documents

Per motius d'il·lustració, he decidit executar aquesta part del codi amb només 10 fitxers, ja que sinó la sortida es fa gegant i no aporta res al resultat. El que es fa per avaluar aquesta segona part es senzill, es carreguen uns quants fitxers xml i el fitxer de les stopwords i es demana a la funció *computeSimilarities*(...) que ens mostri els resultats. Cal notar que es necessari ficar el paràmetre *verbose* a cert per a que es mostrin els resultats.

```
val stopwords = FirstHalf.readFile("stopwordscat-utf8.txt").split(" +").toList
val files = Main.openFiles("wiki-xml-2ww5k", "", ".xml").take(10)
val tstart = System.nanoTime
val nActors = 4
SecondHalf.MapReduceTfIdf.computeSimilarities(files.toList, stopwords, nActors, nActors,
    true)
println("Temps total: " + (System.nanoTime-tstart)/Math.pow(10,9))
```

El resultat de l'execució d'aquest codi és el següent.

```
Calculs iniciats...
TFs calculats!
DFs calculats. Ara falten fer els IDFs
TF_IDFs calculats! Comparem els fitxers!
Resultats del calcul de similaritat entre documents:
Els documents 1000194.xml i 1000231.xml tenen una similaritat del 2.7105183847285925%
Els documents 1000216.xml i 1000098.xml tenen una similaritat del 3.5863902633912086%
Els documents 1000192.xml i 1000152.xml tenen una similaritat del 9.048518813872729%
Els documents 1000192.xml i 1000137.xml tenen una similaritat del 0.9506912909177908%
Els documents 1000194.xml i 1000098.xml tenen una similaritat del 6.286228311394698%
Els documents 1000192.xml i 1000194.xml tenen una similaritat del 30.354524241556334%
Els documents 1000192.xml i 1000231.xml tenen una similaritat del 2.1622333039955226%
Els documents 1000222.xml i 1000098.xml tenen una similaritat del 1.8129191134207028%
Els documents 1000216.xml i 1000231.xml tenen una similaritat del 4.039229826082441%
Els documents 1000192.xml i 1000098.xml tenen una similaritat del 6.154431619715797%
Els documents 1000222.xml i 1000231.xml tenen una similaritat del 2.2611947064760543%
Els documents 1000205.xml i 1000194.xml tenen una similaritat del 2.3590273688580092%
Els documents 1000091.xml i 1000098.xml tenen una similaritat del 4.549167451146728%
Els documents 1000222.xml i 1000152.xml tenen una similaritat del 2.286008959079067%
Els documents 1000216.xml i 1000091.xml tenen una similaritat del 1.0190080306388805%
Els documents 1000098.xml i 1000152.xml tenen una similaritat del 7.374575382396653%
Els documents 1000137.xml i 1000098.xml tenen una similaritat del 0.9762705925831764%
Els documents 1000222.xml i 1000137.xml tenen una similaritat del 0.3890551577815524%
Els documents 1000192.xml i 1000205.xml tenen una similaritat del 2.3186877020865224%
Els documents 1000194.xml i 1000091.xml tenen una similaritat del 8.173194184940954%
Els documents 1000205.xml i 1000216.xml tenen una similaritat del 9.941558511750246%
Els documents 1000222.xml i 1000091.xml tenen una similaritat del 0.6538497186093499%
Els documents 1000192.xml i 1000091.xml tenen una similaritat del 12.660683721068859%
Els documents 1000098.xml i 1000231.xml tenen una similaritat del 1.64416306744753%
Els documents 1000194.xml i 1000137.xml tenen una similaritat del 0.9904819856499192%
Els documents 1000091.xml i 1000231.xml tenen una similaritat del 1.7606972026726193%
Els documents 1000222.xml i 1000192.xml tenen una similaritat del 1.070365654429978%
Els documents 1000222.xml i 1000216.xml tenen una similaritat del 4.486683906277028%
Els documents 1000222.xml i 1000205.xml tenen una similaritat del 0.24647429195418757%
Els documents 1000137.xml i 1000152.xml tenen una similaritat del 0.6577451732054265%
Els documents 1000192.xml i 1000216.xml tenen una similaritat del 1.26877648883606%
Els documents 1000205.xml i 1000091.xml tenen una similaritat del 3.1928936430969044%
Els documents 1000216.xml i 1000152.xml tenen una similaritat del 4.360487047080797%
Els documents 1000205.xml i 1000137.xml tenen una similaritat del 0.3596809717214933%
Els documents 1000194.xml i 1000152.xml tenen una similaritat del 11.138064088885491%
Els documents 1000152.xml i 1000231.xml tenen una similaritat del 1.2590362720544197%
Els documents 1000137.xml i 1000091.xml tenen una similaritat del 0.757480173022309%
Els documents 1000222.xml i 1000194.xml tenen una similaritat del 0.43956683341259095%
Els documents 1000205.xml i 1000231.xml tenen una similaritat del 1.7031060869542374%
Els documents 1000205.xml i 1000098.xml tenen una similaritat del 2.622678833225558%
Els documents 1000091.xml i 1000152.xml tenen una similaritat del 2.308209183997995%
Els documents 1000137.xml i 1000231.xml tenen una similaritat del 0.2018324523935682%
Els documents 1000137.xml i 1000216.xml tenen una similaritat del 0.5311351152307364%
Els documents 1000194.xml i 1000216.xml tenen una similaritat del 1.293387598506396%
Els documents 1000205.xml i 1000152.xml tenen una similaritat del 3.6268341345122526%
Calculs finalitzats. Temps total: 1.6510618
Temps total: 1.6636222
```

Resultats segons els llindars de similitud considerats

Per comparar els resultats amb diferents llindars hem agafat un nombre de fitxers que contingués suficients documents perquè hi haguessin parelles similars però alhora no massa gran perquè el càlcul no tardés molt. Hem decidit agafar 300 documents.

La següent taula mostra el nombre de parelles de documents que obtenim que compleixin cada restricció amb diferents llindars.

Llindar	Parells de pàgines similars no interrelacionades	Parells de pàgines no similars interrelacionades
0.1	127	30849
0.2	24	31015
0.3	7	31047
0.4	4	31060
0.5	2	31068

A continuació es mostra la sortida de les execucions amb els diferents llindars:

Llindar = 0.1

10 primeres parelles de pàgines similars que no es referencien una a l'altra:
William Sholto Douglas, Otto Hoffmann von Waldau ->0.1288872009766826
Castell japonès, Ninja ->0.24025948403874114
William Sholto Douglas, Herbert Otto Gille ->0.10360432690191622
Nantes, William Sholto Douglas ->0.10875873369929959
William Sholto Douglas, Orde Virtuti Militari ->0.16418468360425356
Copa Volpi per la millor interpretació masculina, Christopher Lee ->0.18328534652636125
Ofensiva de Prússia Oriental, Front Oriental de la Segona Guerra Mundial ->0.3728678286341218
William Sholto Douglas, Charles Elwood Yeager ->0.1392076132407418
Andrew Browne Cunningham, Aleksandr Ivànovitx Pokrixkin ->0.11123539067004148
Messerschmitt Me 262, Charles Elwood Yeager ->0.11234735736587806

10 primeres parelles de pàgines que es referencien però no són similars:
Ruth Benedict, Simfonia núm. 8 (Mahler) ->0.009823870046741895
Smith, Història de l'Orient Mitjà ->0.005021916293628367
Sempre en Galiza, Enginyeria inversa ->0.0015703616271788405
Història de Mali, Palau Reial de Caserta ->0.005265578590939846
Neonazisme, Corbeta ->0.0029017094460869566
Memòries d'una geisha (pel·lícula), Reagrupament del Poble Francès ->0.005632770711926015
Rebecca Clarke, Superheroi ->0.005630297770149071
Federació Luterana Mundial, Gran Purga ->0.0017906154961118467
Tres estudis per a figures al peu d'una crucifixió, Història de Kosovo ->0.004374072163271377
Sivaix, Monarquia ->0.00543964030225733

Llindar = 0.2

10 primeres parelles de pàgines similars que no es referencien una a l'altra:
Castell japonès, Ninja ->0.24025948403874114
Ofensiva de Prússia Oriental, Front Oriental de la Segona Guerra Mundial ->0.3728678286341218
William Sholto Douglas, Galeazzo Ciano ->0.23427707057165204
William Sholto Douglas, William Duthie Morgan ->0.22460002270186138
Romania durant la Segona Guerra Mundial, Carles II de Romania ->0.4688662370901408

Andrew Browne Cunningham, HMS Illustrious (87) ->0.21220274194748687
William Sholto Douglas, Andrew McPherson ->0.24777401361516543
Aleksei Antónov, Vasili Marguèlov ->0.2175658681827359
Medalla dels Treballadors Distingits, Orde Virtuti Militari ->0.22175752010250174
Castell japonès, Castell de Malbork ->0.20047780310800487

10 primeres parelles de pàgines que es referencien però no són similars:
Ruth Benedict, Simfonia núm. 8 (Mahler) ->0.009823870046741895
Smith, Història de l'Orient Mitjà ->0.005021916293628367
Sempre en Galiza, Enginyeria inversa ->0.0015703616271788405
Història de Mali, Palau Reial de Caserta ->0.005265578590939846
Neonazisme, Corbeta ->0.0029017094460869566
Memòries d'una geisha (pel·lícula), Reagrupament del Poble Francès ->0.005632770711926015
Rebecca Clarke, Superheroi ->0.005630297770149071
Federació Luterana Mundial, Gran Purga ->0.0017906154961118467
Tres estudis per a figures al peu d'una crucifixió, Història de Kosovo ->0.004374072163271377
Sivaix, Monarquia ->0.00543964030225733

Llindar = 0.3

10 primeres parelles de pàgines similars que no es referencien una a l'altra:
Ofensiva de Prússia Oriental, Front Oriental de la Segona Guerra Mundial ->0.3728678286341218
Romania durant la Segona Guerra Mundial, Carles II de Romania ->0.4688662370901408
Operació Bagration, Ofensiva de Prússia Oriental ->0.4248229547539587
Batalla del Cap Matapan, HMS Illustrious (87) ->0.31551579001078023
Història de l'Argentina, Argentina ->0.5759558134798274
William Sholto Douglas, James O'Meara ->0.30860727097192947
Fußball-Club Bayern München, Cruzeiro Esporte Clube ->0.5139419149640321

10 primeres parelles de pàgines que es referencien però no són similars:
Ruth Benedict, Simfonia núm. 8 (Mahler) ->0.009823870046741895
Smith, Història de l'Orient Mitjà ->0.005021916293628367
Sempre en Galiza, Enginyeria inversa ->0.0015703616271788405
Història de Mali, Palau Reial de Caserta ->0.005265578590939846
Neonazisme, Corbeta ->0.0029017094460869566
Memòries d'una geisha (pel·lícula), Reagrupament del Poble Francès ->0.005632770711926015
Rebecca Clarke, Superheroi ->0.005630297770149071
Federació Luterana Mundial, Gran Purga ->0.0017906154961118467
Tres estudis per a figures al peu d'una crucifixió, Història de Kosovo ->0.004374072163271377
Sivaix, Monarquia ->0.00543964030225733

Llindar = 0.4

10 primeres parelles de pàgines similars que no es referencien una a l'altra:
Romania durant la Segona Guerra Mundial, Carles II de Romania ->0.4688662370901408
Operació Bagration, Ofensiva de Prússia Oriental ->0.4248229547539587
Història de l'Argentina, Argentina ->0.5759558134798274
Fußball-Club Bayern München, Cruzeiro Esporte Clube ->0.5139419149640321

10 primeres parelles de pàgines que es referencien però no són similars:
Ruth Benedict, Simfonia núm. 8 (Mahler) ->0.009823870046741895
Smith, Història de l'Orient Mitjà ->0.005021916293628367

Sempre en Galiza, Enginyeria inversa ->0.0015703616271788405
Història de Mali, Palau Reial de Caserta ->0.005265578590939846
Neonazisme, Corbeta ->0.0029017094460869566
Memòries d'una geisha (pel·lícula), Reagrupament del Poble Francès ->0.005632770711926015
Rebecca Clarke, Superheroi ->0.005630297770149071
Federació Luterana Mundial, Gran Purga ->0.0017906154961118467
Tres estudis per a figures al peu d'una crucifixió, Història de Kosovo ->0.004374072163271377
Sivaix, Monarquia ->0.00543964030225733

Llindar = 0.5

10 primeres parelles de pàgines similars que no es referencien una a l'altra:
Història de l'Argentina, Argentina ->0.5759558134798274
Fußball-Club Bayern München, Cruzeiro Esporte Clube ->0.5139419149640321

10 primeres parelles de pàgines que es referencien però no són similars:
Ruth Benedict, Simfonia núm. 8 (Mahler) ->0.009823870046741895
Smith, Història de l'Orient Mitjà ->0.005021916293628367
Sempre en Galiza, Enginyeria inversa ->0.0015703616271788405
Història de Mali, Palau Reial de Caserta ->0.005265578590939846
Neonazisme, Corbeta ->0.0029017094460869566
Memòries d'una geisha (pel·lícula), Reagrupament del Poble Francès ->0.005632770711926015
Rebecca Clarke, Superheroi ->0.005630297770149071
Federació Luterana Mundial, Gran Purga ->0.0017906154961118467
Tres estudis per a figures al peu d'una crucifixió, Història de Kosovo ->0.004374072163271377
Sivaix, Monarquia ->0.00543964030225733

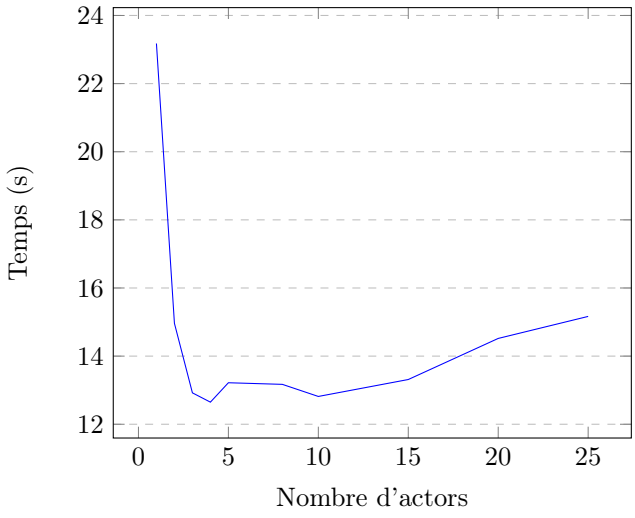
Capítol 5

Rendiment segons el nombre d'actors en el MapReduce

Per fer aquestes proves de rendiment hem usat el mètode SecondHalf.MapReduceTfIdf.computeSimilarities(...) amb els 100 primers documents i hem usat el mateix nombre d'actors tant per el *mapper* com per el *reducer*.

Nombre d'actors	Temps
1	23.177229516
2	14.962622731
3	12.921794558
4	12.648637579
5	13.218281647
8	13.171002857
10	12.816006747
15	13.313055689
20	14.519156598
25	15.165376866

Ho podem veure visualment en el següent gràfic:



En el gràfic veiem que el rendiment millora molt al passar de 1 a 2 actors, lleugerament al posar-ne 3 i molt poc al posar el quart. Al afegir els següents el temps puja lleugerament i a partir

dels 15 actors el temps de còmput augmenta considerablement a mesura que n'anem afegint més. El número òptim sembla ser el 4 i sembla tenir sentit ja que es correspon amb el número de fils que té la màquina.

Les proves de rendiment s'han fet en un intel i5 M430 (2010), 2.26 Ghz, 2 cores, 4 threads, 6 GB de RAM.

Appendices

Main.scala

```
import akka.actor._
import akka.pattern.ask
import scala.concurrent._
import scala.concurrent.duration._
import akka.util.Timeout
import scala.language.postfixOps
import scala.math


/* This object contains all the code of the first part of the hand in.
 */
object FirstHalf {

  def main() = {

    //load stopwords
    val english_stopwords = readFile("test/english-stop.txt").split(" +").toList

    //load pg11 and pg11-net
    val pg11 = readFile("test/pg11.txt")
    val pg11_net = readFile("test/pg11-net.txt")

    //load all the files that will be evaluated with cosinesim()
    val fileSet = new java.io.File("test").listFiles.filter((x) =>
        x.getName.startsWith("pg") && x.getName.endsWith(".txt")).toSet


    /*
     *                    Evaluating freq() function
     */

    Statistics.topN(freq(pg11).sortWith(moreFrequent), 10)

    /*
     *                    Evaluating nonstopfreq() function
     */

    Statistics.topN(nonstopfreq(pg11, english_stopwords).sortWith(moreFrequent), 10)

    /*
     *                    Evaluating paraulesfreqfreq() function
     */

    Statistics.paraulafreqfreqStats(paraulafreqfreq(pg11).sortWith(_._1 < _._1), 10, 5)

     /*
     *                    Evaluating ngrams() function
     */

    Statistics.topN(ngramsfreq(pg11, 3).sortWith(moreFrequent), 10)

    /*
     *                    Evaluating cosinesim() function
     */
```

```
  println(s"Similitud cosinus entre totes les permutacions dels fitxers:
      \n''${fileSet.mkString("'',''")}''")
  println("| ")

  val t0 = System.nanoTime

  for(pair <- fileSet.subsets(2)){
    val t1 = System.nanoTime
    val file_1 = pair.head
    val file_2 = pair.tail.head
    val file_1_str = readFile(file_1.getAbsolutePath)
    val file_2_str = readFile(file_2.getAbsolutePath)
    print("| ")
    Statistics.cosineSimStat(file_1.getName, file_2.getName, cosinesim(file_1_str,
        file_2_str, english_stopwords))
    println("| Calculada en temps: " + (System.nanoTime-t1)/Math.pow(10,9) + " segons")
  }

  println("Temps total per calcular totes les comparacions: " +
      (System.nanoTime-t0)/Math.pow(10,9) + " segons")

}



/*
 * This object contains the "fancy output" functions to evaluate the program results
 */
object Statistics{

  /*
   * Given a list of tuples (_ Int), it prints the tuple in a fancy way and lists the
       division between _._2 and the total sum of _._2 of all the tuples on the list
   * In our case, we use this function to print the list of frequencies of certain
       words in a file, being _._2 the absolute count of occurrences of this word.
   * @param frequencyList List of tuples (_, Int)
   * @param n Number of elements to be shown
   */
  def topN(freqencyList: List[(_, Int)], n: Int) = {
    val nWords = freqencyList.foldLeft(0) { (total, actual) => total + actual._2 }
    val nDiffWords = freqencyList length;
    println("N Words: " + nWords + " Diferent: " + nDiffWords)
    printf("%-30s %-11s %-10s\n", "Words", "ocurrences", "frequency")
    for(r <- freqencyList.slice(0,n)) printf("%-30s %-11d %-10.7f%%\n", r._1, r._2,
        (r._2.toFloat/nWords)*100) //println(r._1 + "   " + r._2 + " " +
        (r._2.toFloat/nWords)*100)
    println()
  }

  /*  Given a frequency list "List[(Int, Int)]", a nMost and a nLeast number, it shows
      (in a fancy way) the first nMost and the last nLeast elements of the list
   *  @param frequenctList The frequency list
   *  @param nMost Number of top elements to show
   *  @param nLeast Number of bottom elements to show
   */
  def paraulafreqfreqStats(frequencyList: List[(Int, Int)], nMost: Int, nLeast: Int) = {
    println("Les " + nMost + " frequencies mes frequents:")
    for(elem <- frequencyList.slice(0, nMost))
```

```
      println(elem._2 + " paraules apareixen " + elem._1)
    println("Les " + nLeast + " frequencies menys frequents:")
    for(elem <- frequencyList.slice(frequencyList.length-nLeast, frequencyList.length))
      println(elem._2 + " paraules apareixen " + elem._1 + " vegades")
    println()
  }

  /*   Given two strings (representing file names) and a number (representing its
       cosinesim), prints the explaination of how this three values correlate to each
       other
   *   @param s1 First string
   *   @param s2 Second string
   *   @param cosinesim The number
   */
  def cosineSimStat(s1: String, s2: String, cosinesim: Double) = {
    printf("La similitud cosinus entre %s i %s es de %10.10f\n", s1, s2, cosinesim)
  }
}



/*  Given two tuples @p x and @p y, evaluates to true if the second element of the y is
     smaller than the second element of x. On draw, it reverses
 *   the criteria and evaluates to true if the first element of x is smaller thant the
      first element of y.
 *   @param x First tuple
 *   @param y Second tuple
 */
def moreFrequent[A <% Ordered[A],B <% Ordered[B]](x: (A, B), y: (A, B)): Boolean =
  x._2 > y._2 || (x._2 == y._2 && x._1 < y._1)

/*  Given a character, evaluates to true if the character meets our criteria of
     acceptable character not to be filtered when reading a file.
 *   @param c The character to be evaluated
 */
def acceptableChar(c: Char): Boolean = c.isLetter || c == ' '

/*  Given a filename, produces a representative string that only contains lower case
     characters and spaces
 *   @param filename the file path of the file to be readed
 *
 *   @pre File must exist, otherwise will throw an exception
 */
def readFile(filename : String) : String = {
 val source = scala.io.Source.fromFile(filename, "UTF-8")
    val str = try source.map(c => if(acceptableChar(c)) c else ' ').mkString finally
        source.close()
    str.toLowerCase.trim
}

/* Given a string @p s, evaluates as a list of absolute frequencies of the words
     (substrings spaced by white spaces ' ') that @p s contains.
 * More specifically, it evaluates as a list of pairs (String, Int), being the String a
      word of @p s and Int being its absolute frequency
 * @param s A string
 */
def freq (s : String) = ngramsfreq(s, 1)
```

```
/* Given a string @p s and a list of excluded words @stopwords, evaluates as the
    absolute frequencies of the words (substrings spaced by white spaces ' ') that @p
    s contains and @p stopwords does not
 * More specifically, it evaluates as a list of pairs (String, Int), being the String a
     word of @p s but not a word of @p stopwords, and Int being its absolute frequency
 * @param s A string
 * @param stopwords A list of strings
 */
def nonstopfreq (s: String, stopwords: List[String]) =
  freq(s).filterNot(x => stopwords.contains(x._1))

/*  Given a string representing a document, it computes the frequencies of the
    frequencies of its words
 *  @param s String representing a document (string with white spaces)
 */
def paraulafreqfreq(s: String): List[(Int, Int)] = {
  val freqfreqMap: collection.mutable.Map[Int, Int] = collection.mutable.Map()
      withDefaultValue(0)
  val frequencies = freq(s)
  for(wordFreq <- frequencies) freqfreqMap(wordFreq._2) += 1

  freqfreqMap.toList.sortWith((x,y) => x._2 > y._2 || (x._2 == y._2 && x._1 < y._1))
}

/*  Given a string representing a document (string with white spaces), it computes the
    frequency of all the ngrams of size n on that document
 *  @param s String representing the document
 *  @param n Length of the ngrams
 */
def ngramsfreq(s: String, n: Int): List[(String,Int)] = {
  val wordMap: collection.mutable.Map[String, Int] = collection.mutable.Map()
      withDefaultValue(0);

  for ( i <- s.split(" +").sliding(n,1) ) wordMap(i.mkString(" ")) += 1 //counting words

  wordMap.toList
}

/*  Computes the euclidean norm of a vector (of Double values)
 *  @param v the Vector
 */
def euclidean_norm(v: Iterable[Double]) = {
  Math.sqrt( v.map( x => x*x ).reduceLeft( _ + _ ) )
}

/*  Given two strings representing two filtered documents, and a list of words that
    will be filtered, it computes its cosine similarity
 *  @param s1 First document
 *  @param s2 Second document
 *  @param stopwords List of words that will be discarded
 */
def cosinesim(s1: String, s2: String, stopwords: List[String]): Double = {
  val freq1 = nonstopfreq(s1, stopwords).toMap;
  val freq2 = nonstopfreq(s2, stopwords).toMap;
  val freq1max = freq1.values.max
  val freq2max = freq2.values.max

  //normalizing vectors
```

```
    val freq1norm = freq1.map{ case (key, value) => (key, value.toDouble/freq1max) }
    val freq2norm = freq2.map{ case (key, value) => (key, value.toDouble/freq2max) }

    //aligning vectors
    val smv1 = freq2norm.map({ case (key, value) => (key, 0.0)}) ++ freq1norm
    val smv2 = freq1norm.map({ case (key, value) => (key, 0.0)}) ++ freq2norm

    //simplify vectors
    val smv1_vec = smv1.values.map(x => x.asInstanceOf[Double])
    val smv2_vec = smv2.values.map(x => x.asInstanceOf[Double])

    //compute cosinesim
    val term = smv1_vec.zip(smv2_vec).map(x => x._1 * x._2).reduceLeft( _ + _ )

    term / (euclidean_norm(smv1_vec) * euclidean_norm(smv2_vec))
  }

}

object SecondHalf {

  /* This object contains all the code that is used to compute similarities between all
      elements in a list of files using MapReduce pattern
   */
  object MapReduceTfIdf{

    /*  First mapping function. Given a file name (not the path) and a tuple (File_path,
        List[stopwords])
     *  generates all the pairs (file name, (word, 1)). This will let us reduce the list
         to a list of ocurrences (frequency)
     *  @param file_name The file name
     *  @param file Tuple of (filePath, List[stopwords])
     */
    def mapping1(file_name: String, file: (String, List[String])): List[(String, (String,
        Double))] = {
      val wordList = tractaxmldoc.readXMLFile(file._1).split("
          +").toList.filterNot(file._2.contains(_))

      val x = (for(word <- wordList) yield (file_name, (word, 1.toDouble)))
      x
    }

    //key-> Filename, values-> list of (Word, count)
    /*  Given a file name and a list of tuples (Word, 1) (word occurrences) of that
        file, reduces the tuples with the same first value to a tuple (Word, n_ocurrences)
     *  @param key Name of the file
     *  @param values List of all occurrences of the words
     */
    def reducing1(key: String, values: List[(String, Double)]): List[(String, Double)] = {
      val res = for( (word, count_list) <- values.groupBy(_._1).toList )
        //For every pair of word and list of counts, add up its counts
        yield (word, count_list.map( {case (_, count) => count } ).reduceLeft( _ + _))

      val max_freq = res.map(_._2).max

      res.sortWith(FirstHalf.moreFrequent).map(x => (x._1, x._2/max_freq))
    }
```

```
/*  Given a file name and its word counts, unfolds all its tuples to the reverse
    (Word, File) tuple.
 *  This will let us count how many documents contain that word and compute the idf
     of that word.
 *  @param file_name Name of that file
 *  @param word_count List of word counts (Word, 1)
 */
def mapping2(file_name: String, word_count: List[(String, Double)]): List[(String,
    String)] = {
  word_count.map(x => (x._1, file_name))
}

/*  Given a file name, and a tuple containing the List of word counts and a Map with
    (at least) the idf of all words in that file
 *  yields all the tuples (file name, (Word, tfidf)) for that list of word counts.
     This will let us compute the space model vector
 *  of that file
 *  @param file_name Name of the file
 *  @param tf_idf_unfolded Pair containing the frequency counts of that file and the
     idf of all the words in that document set
 */
def mapping3(file_name: String, tf_idf_unfolded: (List[(String, Double)], Map[String,
    Double])): List[(String, (String,Double))] = {
  for (term_freq <- tf_idf_unfolded._1)
    yield ( (file_name, (term_freq._1, term_freq._2 *
        tf_idf_unfolded._2(term_freq._1))))
}

/*  Given two maps representing the frequency counts of the words of two files, it
    computes its cosine similarity
 *  @param m1 First document
 *  @param m2 Second document
 */
def cosinesim2(m1: Map[String, Double], m2: Map[String, Double]): Double = {

  //aligning vectors
  val smv1 = m2.map({ case (key, value) => (key, 0.0)}) ++ m1
  val smv2 = m1.map({ case (key, value) => (key, 0.0)}) ++ m2

  //simplify vectors
  val smv1_vec = smv1.values.map(x => x.asInstanceOf[Double])
  val smv2_vec = smv2.values.map(x => x.asInstanceOf[Double])

  //compute cosinesim
  val term = smv1_vec.zip(smv2_vec).map(x => x._1 * x._2).reduceLeft( _ + _ )

  term / (FirstHalf.euclidean_norm(smv1_vec) * FirstHalf.euclidean_norm(smv2_vec))
}

/*  Given a pair of file names, and its representing space model vectors, computes
    its cosine similarity
 *  @param files Pair of file names
 *  @param tf_idfs Pair of space model vectors
 */
def mapping4(files: (String, String), tf_idfs:(List[(String, Double)], List[(String,
    Double)]) ) = {
  val cosinesim = cosinesim2(tf_idfs._1.toMap, tf_idfs._2.toMap)
```

```
  List((files, cosinesim))
}

/*  Doesn't reduce. It's simply a pass through for the @p second parameter
 *  @param first First parameter
 *  @param second Second parameter
 */
def passThroughtReduce[A](first: Any, second: List[A]) = {
  second
}

/*  Given a list of files, the list of stopwords and the number of mappers and
    reducers desired computes the cosine similarirty between
 *  all possible pair of files excluding the stopwords using the MapReduce pattern.
 *  @param files List of files to be compared
 *  @param stopwords List of words to be filtered before comparing
 *  @param nMappers Number of mapping actors
 *  @param nReducers Number of reducing actors
 *  @param verbose Optional parameter, the method will explain you what is doing at
     each step of its computing spree.
 */
def computeSimilarities(files: List[java.io.File], stopwords: List[String], nMappers:
    Int, nReducers: Int, verbose: Boolean = true) = {

  if (verbose) println("Calculs iniciats...")
  val tstart = System.nanoTime

  implicit val timeout = Timeout(10 days)

  //// 1/4 Counting Words ////

  //From files, generate a list of pairs (file name, (file path, stopwords))
  val input = ( for( file <- files) yield (file.getName, (file.getAbsolutePath,
      stopwords)) ).toList

  //initializing actor system
  val system = ActorSystem("TextAnalizer2")

  //master actor to control MapReduce
  var master = system.actorOf(Props(new MapReduceActor[String, (String,
      List[String]), String, (String, Double)](input, mapping1, reducing1, nMappers,
      nReducers)))

  val futureResponse1 = master ? "start"
  //waiting for master's response
  val tf = Await.result(futureResponse1, timeout.duration).asInstanceOf[Map[String,
      List[(String, Double)]]]

  if (verbose) println("TFs calculats!")

  //stoping old master
  system.stop(master)

  //// 2/4 Computing DF ////

  //creating new master
  master = system.actorOf(Props(new MapReduceActor[String, List[(String, Double)],
      String, String](tf.toList, mapping2, passThroughtReduce[String], nMappers,
```

```
        nReducers)))

    val futureResponse2 = master ? "start"
    //waiting for master's response
    val df = Await.result(futureResponse2, timeout.duration).asInstanceOf[Map[String,
        List[String]]]

    if (verbose) println("DFs calculats. Ara falten fer els IDFs")

    //stoping old master
    system.stop(master)

    //// 3/4 Computing TF_IDF ////

    //Computing the idf of every word
    //this line could be done with MapReduce, but the overhead caused by map and reduce
        actor initialization is not worth the time
    val idf = df.map(x => (x._1, Math.log(tf.size/x._2.length)))

    //preparing input: adding the idf to all input values
    //input: List[File -> ( List[(Word, dtf)], List[(Word, idf)])]
    val tfIdfInput = tf.map({case (k,v) => (k, (v,idf))})

    //creating new master
    master = system.actorOf(Props(new MapReduceActor[String, (List[(String, Double)],
        Map[String, Double]), String, (String, Double)](tfIdfInput.toList, mapping3,
        passThroughtReduce[(String,Double)], nMappers, nReducers)))

    val futureResponse3 = master ? "start"
    //waiting for master's response
    val tf_idf = Await.result(futureResponse3,
        timeout.duration).asInstanceOf[Map[String, List[(String, Double)]]]

    if (verbose) println("TF_IDFs calculats! Comparem els fitxers!")

    //stoping old master
    system.stop(master)

    //// 4/4 Comparing all files ////

    //Preparing input: Creating all paris of files to be compared, with its space model
        vectors included
    //Map[(FileName, FileName) -> (List[(Word, tfidf)], List[(Word, tfidf)])
    val comparisonList = for ( pair <- tf_idf.toSet.subsets(2) ) yield {
      ((pair.head._1, pair.drop(1).head._1), (pair.head._2, pair.drop(1).head._2))
    }

    //creating new master
    master = system.actorOf(Props(new MapReduceActor[(String, String), (List[(String,
        Double)], List[(String, Double)]), (String, String),
        Double](comparisonList.toList, mapping4, passThroughtReduce[Double], nMappers,
        nReducers)))

    val futureResponse4 = master ? "start"
    //waiting for master's response
    val result = Await.result(futureResponse4,
        timeout.duration).asInstanceOf[Map[(String, String), List[Double]]]
```

```
    //final product: Map[(Filename, Filename), cosinesim]
    val finalResult = result.map({case (k,v) => (k, v.apply(0))})

    val tend = System.nanoTime

    //printing the results
    if (verbose){
      println("Resultats del calcul de similaritat entre documents: ")

      for(singleResult <- finalResult){
        println("Els documents " + singleResult._1._1 + " i " + singleResult._1._2 + "
            tenen una similaritat del " + singleResult._2*100 + "%")
      }

      println("Calculs finalitzats. Temps total: " + (tend-tstart)/Math.pow(10,9))
    }

    //Shutdown actor system
    system.shutdown

    finalResult
  }

}


object MapReduceRef {

  def mapping1(title: String, referencedDocs: List[String]): List[(String, String)] = {
    for(doc <- referencedDocs) yield (doc, title) //document, document that references
        the first one
  }

  def reducing1(title: String, docsRefer: List[String]): List[String] = {
    docsRefer.distinct
  }

  /*
  def mapping2(title: String, tuple: (List[String],List[String])): List[(String,
      String)] = {
    val allTitles = tuple._2
    for(titleRef <- allTitles; if(!tuple._1.contains(titleRef))) yield (title, titleRef)
  }

  def reducing2(title: String, docsRefer: List[String]): List[String] = {
    docsRefer
  }*/

  def mapReduceDocumentsNoReferenciats() = {

    val files = Main.openFiles("wiki-xml-2ww5k", "", ".xml").toList.take(100)
    val input = for(file <- files) yield tractaxmldoc.referencies(file)
    val titles = tractaxmldoc.titols(files)

    var system = ActorSystem("DocsReferenciats")
    var master = system.actorOf(Props(new MapReduceActor[String, List[String], String,
        String](input, mapping1, reducing1, 2, 2)))
    implicit val timeout = Timeout(10 days)
```

```
val futureResponse = master ? "start"
val result = Await.result(futureResponse, timeout.duration)
system.stop(master)

/*
val input2 = for(elem <- filterDocuments.toList) yield {
  val temp = elem
  val title = temp._1
  val refs = temp._2
  (title, (refs, titles))
}
master = system.actorOf(Props(new MapReduceActor[String, (List[String],
    List[String]), String, String](input2, mapping2, reducing2, 2, 2)))
val futureResponse2 = master ? "start"
val result2 = Await.result(futureResponse, timeout.duration)

val result2map = result2.asInstanceOf[Map[String,List[String]]]*/

val result1map = result.asInstanceOf[Map[String,List[String]]] //Map[title,
    list[title of the documents that refer to it]]
val filterDocuments = result1map.filter(x => titles.contains(x._1)) //we only keep
    the titles of the documents that we have

val result2 = for(i <- filterDocuments.toList) yield (i._1, titles.filterNot(x =>
    i._2.contains(x) || x.equals(i._2)))
val result2map = result2.toMap
val result2filled = for(title <- titles) yield {
  if (result2map.contains(title))
    (title, result2map(title))
  else
    (title, List())
}
val result2mapfilled = result2filled.toMap

val Llindar: Double = 0.2

val stopwords = FirstHalf.readFile("stopwordscat-utf8.txt").split(" +").toList
val tf_idf = MapReduceTfIdf.computeSimilarities(files, stopwords, 10, 10, false)
val tf_idf2 = tf_idf.filter(x => x._2 > Llindar)

val fileMap = tractaxmldoc.titolsNomfitxer(files).toMap
val tf_idf3 = tf_idf2.filter(x =>
    result2mapfilled(fileMap(x._1._1)).contains(fileMap(x._1._2)) )

println("10 primeres parelles de pagines similars que no es referencien una a
    l'altra")
for(i <- tf_idf3.take(10)) println(i)


val tf_idf4 = tf_idf.filter(x => x._2 < Llindar)
val tf_idf5 = tf_idf4.filterNot(x =>
    result2mapfilled(fileMap(x._1._1)).contains(fileMap(x._1._2)) )

println()
println("10 primeres parelles de pagines similars que no es referencien")
for(i <- tf_idf5.take(10)) println(i)

system.shutdown
```

```
    }
  }

}


object Main extends App {

  /* Given a folder and two strings with the prefix and sufix of the files to be opened
      it returns an array of file handlers
   *  @param folder Name of the folder to search the files in
   *  @param startsWith Prefix of the files to be opened
   *  @param endsWith Sufix of the files to be opened
   */
  def openFiles(folder: String, startsWith: String, endsWith: String):
      Array[java.io.File] = {
    var fileList = new
        java.io.File(folder).listFiles.filter(_.getName.startsWith(startsWith)).filter(_.getName.endsWith(endsW
    fileList
  }

  /*
   * Main method of the App
   */
  override def main(args:Array[String]) = {

    FirstHalf.main()

    println("\n")

    val stopwords = FirstHalf.readFile("stopwordscat-utf8.txt").split(" +").toList
    val files = Main.openFiles("wiki-xml-2ww5k", "", ".xml").take(100)
    SecondHalf.MapReduceTfIdf.computeSimilarities(files.toList, stopwords, 10, 10, true)

    println("\n")

    SecondHalf.MapReduceRef.mapReduceDocumentsNoReferenciats()
  }
}
```

MapReduceActor.scala

```
import akka.actor._

/* Class extending akka Actor containing an implementation of MapReduce design pattern.
 *    To start the algorithm you have to instantiate an actor of this class, and then
      send him the message "start".
 *    Eventually it will compute and return to you the result.
 *    @param input List of pairs key value where the mapping function will be applied
 *    @param mapping Mapping function
 *    @param reducing Reduction function
 *    @param numMappers Number of actors that will be assigned to mapping operations
 *  @param numReducers Number of actors that will be assigned to reducing operations
 */
class MapReduceActor[K, V, K2, V2]
    (
    input:   List[(K, V)],
    mapping: (K, V) => List[(K2, V2)],
    reducing: (K2, List[V2]) => List[V2],
    numMappers: Int,
    numReducers: Int                    ) extends Actor{

  //master - service messages
  case class MapOrder(k: K, v: V, mapping: (K, V) => List[(K2, V2)])
  case class ReduceOrder(k: K2, vlist: List[V2], reducing: (K2, List[V2]) => List[V2])

  //service - master messages
  case class Intermediate(list: List[(K2, V2)])
  case class Reduced(key: K2, list: List[V2])

  val master = self
  var invoker: akka.actor.ActorRef = null

  var intermediates = List[(K2, V2)]()
  var pendingIntermediates: Int = 0

  var result = Map[K2, List[V2]]()
  var pendingReduceds: Int = 0

  //unused
  def cleanActors() = {
    for (child <- context.children){
      context.stop(child)
    }
  }

  def receive = {
    //first call
    case "start" =>
      invoker = sender

      if (input.length > 0) { //if there's something to compute

        val groupSize = if (input.length/numMappers == 0) 1 else input.length/numMappers
        val groups = input.grouped(groupSize)

        var workers: List[akka.actor.ActorRef] = List()
```

```
    //split the load between the actors
    for (group <- groups) {
      val worker = context.actorOf(Props(new Actor {
        def receive = {
          case MapOrder(key, value, mapping) => sender ! Intermediate(mapping(key,
              value))
        }
      }))

      workers ::= worker

      //order actors to map the pairs key value
      for ((key, value) <- group) worker ! MapOrder(key, value, mapping)
    }

    pendingIntermediates = input.length
  }
  else invoker ! Map()

//Between mapping and reducing
case Intermediate(list) =>
  intermediates = list ::: intermediates
  pendingIntermediates -= 1

  if(pendingIntermediates == 0){

    var dict = Map[K2, List[V2]]() withDefault (k => List())

    for ((key, value) <- intermediates)
      dict += (key -> (value :: dict(key)))

    var workers: List[akka.actor.ActorRef] = List()

    val groupSize = if (dict.size/numMappers == 0) 1 else dict.size/numMappers
    val groups = dict.grouped(groupSize)

    //split the load between the actors
    for (group <- groups){
      val worker = context.actorOf(Props(new Actor {
        def receive = {
          case ReduceOrder(key, values, reducing) => sender ! Reduced(key,
              reducing(key, values))
        }
      }))

      workers ::= worker

      //order actors to reduce the values
      for ((key, values) <- group) worker ! ReduceOrder(key, values, reducing)

    }

    pendingReduceds = dict.size
  }

//adding up the results
case Reduced(key, values) =>
```

```
      result += (key -> values)
      pendingReduceds -= 1

      if(pendingReduceds == 0) //if we have finished, send back to invoker the results

        invoker ! result
  }

}
```

referencies.scala

```
import scala.xml.XML
import scala.util.matching.Regex

object tractaxmldoc{

  /* Given a file handler of an xml file returns a tuple with the title and the list of
      references found
   *  @param file The file handler of the xml file
   */
  def referencies(file: java.io.File): (String, List[String]) = {
    val xmlleg=new java.io.InputStreamReader(new java.io.FileInputStream(file), "UTF-8")
     val xmllegg = XML.load(xmlleg)
     val title=(xmllegg \\ "title").text

     val contingut = (xmllegg \\ "text").text
     val ref = new Regex("\\[\\[[^\\]]*\\]\\]")
     val refs=(ref findAllIn contingut).toList

     //The order of the operations is important. There'll be references to pages that we
         don't have
     val kk3 = refs.filterNot(x=> x.contains(':') ||
         x.apply(2)=='#').map(x=>x.substring(2,x.length()-2)).map(x=>x.split("\\|").apply(0)).map(x=>x.split("#"

     (title, kk3)
  }

  /* Given an array of xml file handlers return the titles of the files
   *  @param files The array of xml file handlers
   */
  def titols(files: List[java.io.File]): List[String] =
    for(file <- files) yield {
      val xmlleg=new java.io.InputStreamReader(new java.io.FileInputStream(file), "UTF-8")
     val xmllegg = XML.load(xmlleg)
     val title=(xmllegg \\ "title").text
     title
    }

  /* Given an array of xml file handlers returns a list of tuples with the filename and
      the title
   *  @param files The array of xml file handlers
   */
  def titolsNomfitxer(files: List[java.io.File]): List[(String,String)] =
    for(file <- files) yield {
      val xmlleg=new java.io.InputStreamReader(new java.io.FileInputStream(file), "UTF-8")
```

```
    val xmllegg = XML.load(xmlleg)
    val title=(xmllegg \\ "title").text
    (file.getName, title)
   }

  /* Given an xml file, extracts the contents of the text region and produces a
      representative string that only contains lower case characters and spaces
   *  @param filename The name of the xml file to be read
   */
  def readXMLFile(filename : String) : String = {
    val xmlleg=new java.io.InputStreamReader(new java.io.FileInputStream(filename),
        "UTF-8")
    val xmllegg = XML.load(xmlleg)
    //val titol=(xmllegg \\ "title").text
    val contingut = (xmllegg \\ "text").text

     val str = try contingut.map(c => if(FirstHalf.acceptableChar(c)) c else '
         ').mkString finally xmlleg.close()
     str.toLowerCase.trim.replaceAll(" +", " ")
  }

}
```