Pràctica de programció funcional + orientada a objectes

Similitud entre documents

Ismael El Habri, Lluís Trilla

16 d'octubre de 2018

Índex

1 Codi de la pràctica **5**

1.1 Fitxer SimilitudEntreDocuments.scala 5

1.1.1 Funcions de freqüència 5

1.1.1.1 Funció Freqüència 5

1.1.1.2 Funció de normalització 6

1.1.1.3 Funció de Freqüències sense *StopWords* 6

1.1.1.4 Funció de distribució de paraules 6

1.1.2 Funcions de Comparació 7

1.1.2.1 Funcions auxiliars 7

1.1.2.2 Funció cosinesim 7

1.1.3 Funcions pel MapReduce 8

1.1.3.1 Funció per Llegir fitxers XML 8

1.1.3.2 Llista de fitxers en un directori 8

1.1.4 Funció del Apartat 2 de la Pràctica 9

1.1.4.1 Llegir Fitxers i tractar-los 9

1.1.4.2 Càlcul del vector IDF 10

1.1.4.3 Comparacions tots amb tots 11

1.1.4.4 Tractant referències 12

1.1.4.5 Volcatge dels resultats en un fitxer 13

1.1.5 Main de l'aplicació 15

1.1.5.1 Primera part 15

1.1.5.2 Segona Part 16
1.2 Fitxer MapReduceFramework.scala 16
1.2.1 Idea General 17
1.2.2 Signatura 17
1.2.2.1 Sobre el Tipatge 17
1.2.3 Classes internes 17
1.2.4 Inicialització de valors 18
1.2.5 Definició dels routers de Map i Reduce 18
1.2.6 Funció de receive 18

2 Joc de proves **20**

2.1 Resultats amb els fitxers donats 20
2.1.1 pg11.txt 20
2.1.2 pg11-net.txt 21
2.1.3 pg12.txt 22
2.1.4 pg12-net.txt 24
2.1.5 pg74.txt 25
2.1.6 pg74-net.txt 26
2.1.7 pg2500.txt 27
2.1.8 pg2500-net.txt 28
2.2 Resultat amb Fitxer personalitzat 30
2.2.1 Contingut del fitxer 30
2.2.2 Resultat 30
2.3 Resultats Cosinesim 31
2.3.1 pg-11.txt amb pg11-net.txt 31
2.3.2 pg-11.txt amb pg74.txt 31
2.3.3 pg-12.txt amb pg74.txt 32
2.3.4 pg-12.txt amb pg-12.txt 32

3 Resultats **33**

3.1 Taula de rendiment de MapReduce segons nombre d'actors 33

3.2 Taula d'articles a referenciar segons llindar 33

3.3 Ex.3 34

Capítol 1

Codi de la pràctica

Hem dividit el nostre codi en dos fitxers, SimilitudEntreDocuments.scala i MapReduceFramework.scala.

1.1 Fitxer SimilitudEntreDocuments.scala

Aquest fitxer inclou les següents funcions:

1.1.1 Funcions de freqüència

1.1.1.1 Funció Freqüència

```
//Rebent una String i un enter n dóna com a resultat una llista amb tuples (n-grames,
    freqüència)
def freq(text:String, n:Int):List[(String, Int)] =
  normalitza(text).split(" +").sliding(n).toList
    .map(_.mkString(" ")).groupBy(identity).mapValues(_.length).toList
```

Aquesta funció rep una String amb el contingut d'un fitxer i un enter, i torna una Llista amb tuples n-grames (on n és l'enter donat), Nombre. El nombre és el nombre de vagades que apareix en el fitxer la paraula amb la que va agrupat. En la funció usem la funció que explicarem a continuació, `normalitza`. A part, usem les següents funcions de Scala:

- `split`: separa una string en una llista de strings usant un delimitador donat
- `sliding`: per fer grups de n-elements amb els elements de la llista de Strings.
- `map`: L'usem per convertir cada grup resultant de la funció anterior en Strings.
- `mkString`: Per crear Strings a partir de la llista.
- `groupBy`: S'usa per agrupar els elements d'una llista donada una certa relació, en aquest cas volem agrupar els ideals, això dóna un resultat del tipus `Map[String,List[String]]`.

- `mapValues`: L'usem per reduir els resultats dels valors anteriors en un element de Diccionari, en aquest cas: String -¿ Int.

1.1.1.2 Funció de normalització

```
//Rep una String i la normalitza (canvia tot el que no és lletra per espais i passa la
    string a minúscules)
def normalitza(text:String):String =
  text.map(c=> if(c.isLetter) c else ' ').toLowerCase().trim
```

Funció que per cada element de una String fa un map per canviar tots els elements que no són lletres per espais, passa el resultat del map a minúscules.

De les funcions de Scala no n'usem cap de nova, apart de `trim` que ens treu els espais generats al principi i al final de la String normalitzada.

1.1.1.3 Funció de Freqüències sense *StopWords*

```
//Rep una String i una llista de Strings amb stop words, i fa el vector de frequencies
    filtran les stop words.
def nonStopFreq(text:String, stop:List[String], n:Int):List[(String, Int)] =
  normalitza(text).split(" +").filterNot{a =>
      stop.contains(a)}.sliding(n).toList.map(_.mkString("
      ")).groupBy(identity).mapValues(_.length).toList
```

Funció molt semblant a la de freqüències normals, però en aquest cas, abans del sliding, filtrem les *stopwords* donades.

1.1.1.4 Funció de distribució de paraules

```
//Obtenim les 10 frequències més frequents, i les 5 menys frequents
def paraulafreqfreq(llistaFrequencies:List[(String,Int)]): Unit = {
  val stringFrequencies:String = llistaFrequencies.map(_._2.toString.concat(" ")).mkString
  val freqfreqList = stringFrequencies
    .split(" +").groupBy(identity).mapValues(_.length).toList.sortBy(-_._2)
  println("Les 10 frequencies mes frequents:")
  for(frequencia <- freqfreqList.slice(0,10))
    println(frequencia._2 + " paraules apareixen " + frequencia._1 +" vegades")
  println("Les 5 frequencies menys frequents:")
  for(frequencia <-
      freqfreqList.slice(freqfreqList.length-5,freqfreqList.length).sortBy(-_._2))
    println(frequencia._2 + " paraules apareixen " + frequencia._1 +" vegades")
}
```

Funció on busquem les 10 freqüències més freqüents, i les 5 que ho son menys. Esta implementada aprofitant la mateixa funció freq original. Primer de tot transformem l'estructura de dades que conté les freqüències en una string amb les freqüències separades per espais, i després nomes cal usar la funció freq per tal d'obtenir les freqüències de freqüències

1.1.2 Funcions de Comparació

La única funció per comparar és la funció `cosinesim`, però abans, introduirem les funcions auxiliars utilitzades per fer-lo.

1.1.2.1 Funcions auxiliars

```
//Rep dos vectors de frequencies absolutes i les converteix a tf.
def freqAtf(llistaFreq:List[(String, Int)]):List[(String, Double)] = {
  val mesfrequent = llistaFreq.maxBy(_._2)._2
  llistaFreq.map{a=> (a._1, a._2.toDouble/mesfrequent)}
}

//Retorna la paraula no-stop més frequent del text introduït, junt amb la seva freqüència
def mesFrequent(text:String, stop:List[String], n:Int) = nonStopFreq(text, stop,
    n).maxBy(_._2)

//Busquem les paraules q no tenim a txt1 de txt2 i les afegim amb frequencia = 0 a txt1
    Al final ordenem alfabeticament, per tenir el mateix ordre en els dos vectors!
def alinearVector(aAlinear:List[(String, Double)], suport:List[(String,
    Double)]):List[(String, Double)] ={
  val aAlinearMap = aAlinear.toMap
  (aAlinear ::: (for (b<-suport if !aAlinearMap.contains(b._1)) yield (b _1, 0.0)))
      sortBy(_._1)
}
```

1. `freqAtf`: passa les freqüències absolutes a freqüència *tf*.
2. `mesFrequent`: ens dóna l'element més freqüent de la llista de freqüències.
3. `alinearVector`: Donat un vector a alinear, i un vector amb el qual s'ha de alinear, alinea el vector a alinear.

1.1.2.2 Funció cosinesim

```
//Rep dos vectors amb les paraules i les seves freqüencies tf, i retorna la semblança
    entre aquests dos fitxers.
def cosinesim (txt1:List[(String,Double)], txt2:List[(String,Double)]):Double =
  (for ((a, b) <- alinearVector(txt1,txt2) zip alinearVector(txt2,txt1)) yield a._2 *
      b._2).foldLeft(0.0)(_ + _)/(sqrt(txt1.foldLeft(0.0){(a,b)=> a+(b._2*b._2)}) *
      sqrt(txt2.foldLeft(0.0){(a,b)=> a+(b._2*b._2)}) )
```

Sent la fórmula de simil·litud:

$$sim(a,b) = \frac{a \cdot b}{sqrt\sum_{i=1}^{m} a[i]^2 \cdot sqrt\sum_{i=1}^{m} b[i]^2}$$

El cosinesim ha de fer la divisió del producte escalar dels dos vectors alineats entre el resultat de la multiplicació de la arrel de la suma de cada *tf* dels dos vectors de frequencies.

La funció dona per suposat que la freqüència donada és freqüència *tf*.

1.1.3 Funcions pel MapReduce

1.1.3.1 Funció per Llegir fitxers XML

```
//Rep el nom de un docoment de la wiki, el llegeix i el filtra, resultant en el nom de
    l'article, el contingut d'aquest i una llista de referències cap a altres articles
def tractaXMLdoc(docName:String): (String, String, List[String]) = {
  val xmlleg=new java.io.InputStreamReader(new java.io.FileInputStream(docName), "UTF-8")
  val xmllegg = XML.load(xmlleg)
  // obtinc el titol
  val titol=(xmllegg \\ "title").text
  // obtinc el contingut de la pàgina
  val contingut = (xmllegg \\ "text").text

  // identifico referències
  val refs=(new Regex("\\[\\[[^\\]]*\\]\\]") findAllIn contingut).toList
  // elimino les que tenen : (fitxers) i # (referencies internes)
  val kk = refs.filterNot(x=> x.contains(':') || x.contains('#')).map(_.takeWhile(_!='|'))
  (normalitza(titol), contingut, kk.map(normalitza).distinct)
}
```

Funció basada en el programa Scala donat pel professor, modificada per tal de eliminar referencies internes (les que contenen el caràcter '#'), i treure la part de les referencies que parla del apartat del article referit (a partir del caràcter '|').

1.1.3.2 Llista de fitxers en un directori

```
//donada una string de directori, retorna la llista de fitxers que conté
def llistaFitxers(dir: String):List[String] = new
    File(dir).listFiles.filter(_.isFile).toList.map{a => dir + "/" + a.getName}
```

A partir del nom d'un directori, ens torna una llista amb els paths dels fitxers del directori. Per fer-ho usem funcions del Java.

1.1.4 Funció del Apartat 2 de la Pràctica

Tenim tota la segona part de la pràctica en la següent funció, que rep una n corresponent als n-grames que volem utilitzar per fer les funcions, el systemActor per poder iniciar actors, i el nombre de workers que volem que el MapReduce utilitzi:

```
def calcularSimilituds(n: Int, system: ActorSystem, numWorkers: Int)= {
  type Fitxer = (String,(List[(String,Double)],List[String]))

  val fitxers = llistaFitxers(DirectoriFitxers)//input
  ...
}
```

1.1.4.1 Llegir Fitxers i tractar-los

Aquesta primera part definim un alies pels fitxers, i aconseguim la llista de fitxers a tractar.

A continuació vindria un MapReduce per a la lectura de fitxers. Aquest treballarà així:

1. El map s'encarregarà de llegir cada fitxer de disc, convertint-lo en un element de tipus (String, String, List[String]), sent aquests el títol del article, el contingut d'aquest, i la seva llista de referències a altres articles.
2. El reduce calcularà la freqüència *tf* de cada fitxer.

```
//funció que llegeix i tracta un fitxer resultat en el títol, el contingut, i una llista
    de referències.
  def mapFunctionReadFiles(file:String):(String, (String, List[String])) = {
    val valors = tractaXMLdoc(file)
    (valors._1,(valors._2,valors._3))
  }

  //funció que a partir d'un fitxer en format(Títol, (Contingut, Referencies), calcula
      les frequencies TF en el contingut
  def reduceFunctionReadFiles(fileContent:(String, (String, List[String]))):Fitxer =
    (fileContent._1, (freqAtf(nonStopFreq(fileContent._2._1,stopWords,1)),
        fileContent._2._2))


  //Fem l'actor del MapReduce
  val act = system.actorOf(Props(
    new MapReduceFramework[String, String, (String,List[String]),String,
        (String,List[String]),String, (List[(String, Double)], List[String])](
      {f:String => List(mapFunctionReadFiles(f))},
      {f:List[(String, (String, List[String]))]=>f},
      {(f:String, s:(String, List[String])) => List(reduceFunctionReadFiles((f,s)))},
```

```
    numWorkers,numWorkers,fitxers)
  ))

  //L'hi enviem el missatge de inicialització al MapReduce, després esperem el resultat,
      usant un pattern.
  implicit val timeout = Timeout(12000,TimeUnit.SECONDS)
  val futur = act ? Iniciar()
  val diccionariFitxers =
      Await.result(futur,timeout.duration).asInstanceOf[mutable.Map[String,
      (List[(String, Double)], List[String])]]
  //parem l'actor
  act ! PoisonPill
```

1.1.4.2 Càlcul del vector IDF

Després la funció faria un mapReduce per calcular el vector *idf*. Aquest funcionaria així:

- el map ens genera tuples (paraules,valors) amb cada paraula diferent de cada fitxer, inicialitzat a 1.
- una funció intermèdia que ens agrupa les paraules iguals
- el reduce agafa cada grup d'aquests, i en calcula el valor *idf*

```
//funció que rep un fitxer i resulta en una Llista de parelles de paraules diferents en
    el fitxer, i el número 1
def mapFunctionIDF(fitxer:Fitxer):List[(String,Double)] =
  (fitxer._2)._1.map{x=>(x._1,1.0)}

//funció que rep una parella amb una Paraules i Llista de Paraules(iguals) i nombres,
// la funció conta la llargada de la funció
def reduceFunctionIDF(dades:(String,List[(String,Double)])):(String,Double) =
  (dades._1, log10(nombreFitxers/ dades._2.foldLeft(0){ (a, _)=>a+1}))

//funció que rep una Llista de paraules inicialitzades a 1,
// la funció agrupa les paraules iguals en llistes de parelles Paraula, grup de
    (paraules, nombre)
def funcioIntermitjaIDF(dades:List[(String,Double)]):List[(String,List[(String,Double)])]
    =
  dades.groupBy(_._1).toList


//Fem l'actor del MapReduce
val act2 = system.actorOf(Props (new MapReduceFramework[Fitxer,
  String,Double,String,List[(String,Double)],String,Double](
  {f=>mapFunctionIDF(f)},
  {f=>funcioIntermitjaIDF(f)},
  {(f:String,s:List[(String,Double)])=>List(reduceFunctionIDF((f,s)))},
```

```
  numWorkers,numWorkers,diccionariFitxers.toList
)))

//L'hi enviem el missatge de inicialització al MapReduce, després esperem el resultat,
    usant un pattern.
val futur2 = act2 ? Iniciar()
val diccionariIDF =
    Await.result(futur2,timeout.duration).asInstanceOf[mutable.Map[String, Double]]
//parem l'actor
act2 ! PoisonPill
```

1.1.4.3 Comparacions tots amb tots

En aquest punt tocaria fer la comparació dels fitxers tots amb tots. Això òbviament amb un MapReduce:

- el map s'encarregarà de aplicar el *idf* de cada paraula als vectors *tf*.
- una funció intermèdia que s'encarregarà de generar cada possible comparació evitant simetries.
- el reduce farà les comparacions de cada fitxer amb els que li toquin.

```
//funció que multplica el IDF corresponent per a cada tf de cada paraula, formant el
    vector TF_IDF de un fitxer donat.
def mapComparacio(fitxer:Fitxer):Fitxer =
  (fitxer._1, (fitxer._2._1.map{f=> (f._1,f._2 * diccionariIDF(f._1))}, fitxer._2._2))

//funció que donada una llista de fitxers, per cada fitxer, genera una Llista amb tots
    els fitxers següents
def generarComparacions(fitxers:List[Fitxer]):List[(Fitxer,List[Fitxer])] = {
  if (fitxers.isEmpty) Nil:List[(Fitxer,List[Fitxer])]
  else {
    val fitxersSenseCap = fitxers.tail
    List((fitxers.head, fitxersSenseCap)) ::: generarComparacions(fitxersSenseCap)
  }
}

//funció que donada una comparació d'un fitxer amb una llista de fitxers,
// resulta en una tupla amb el títol del fitxer i una Llista de parelles amb títols de
    fitxers i resultats de comparacions
def reduceComparacio(comparacions:(Fitxer,List[Fitxer])):(String, List[(String, Double)])
    =
  (comparacions._1._1, for(f <- comparacions._2) yield (f._1,
      cosinesim(comparacions._1._2._1,f._2._1)))


//Fem l'actor del MapReduce
val act3 = system.actorOf(Props (new MapReduceFramework[Fitxer,
  String,(List[(String,Double)],List[String]),
```

```
  Fitxer, List[Fitxer],
  String, List[(String, Double)]](
  {f=>List(mapComparacio(f))},
  {f=>generarComparacions(f)},
  {(f:Fitxer,s:List[Fitxer])=>List(reduceComparacio((f,s)))},
  numWorkers,numWorkers,diccionariFitxers.toList
)))
//L'hi enviem el missatge de inicialització al MapReduce, després esperem el resultat,
    usant un pattern.
val futur3 = act3 ? Iniciar()
val resultatComparacions =
    Await.result(futur3,timeout.duration).asInstanceOf[mutable.Map[String, List[(String,
    Double)]]].toList.sortBy(-_._2.length)
//parem l'actor
act3 ! PoisonPill
```

Amb això el primer apartat de la segona part de la Pràctica estaria acabat, faltant el segon.

1.1.4.4 Tractant referències

El segon apartat consisteix en buscar els articles amb similituds per sobre d'un llindar però que no es referenciïn i fitxers que es referenciïn però no estiguin per sobre de cert altre llindar. Les dos les hem fet amb mètodes semblants, Un MapReduce per cada un que no fa res en el Reduce.

- el map s'encarrega de filtrar els elements per sobre `||` sota d'un llindar que no es `||` sí es referenciïn.
- no hi ha cap funció de tractament intermedi.
- no hi ha cap funció de reducció

Primer doncs, el primer cas, sent aquest el cas en què busquem parelles per sobre de cert llindar que no es referenciïn:

```
//donada llista referencies
  //map: eliminar els que no superin cert llindar, despres eliminar els no referenciats
  def mapObtenirNoRefs(fitxer:(String, List[(String, Double)])):(String, List[(String,
      Double)]) ={
    (fitxer._1,fitxer._2.filter(_._2>LlindarNoReferenciats).filter{
      f => !diccionariFitxers(fitxer._1)._2.contains(f._1) &&
          !diccionariFitxers(f._1)._2.contains(fitxer._1)
    })
  }
  //Fem l'actor del MapReduce
  val act4 = system.actorOf(Props (new MapReduceFramework[
    (String, List[(String, Double)]),
    String,List[(String, Double)],
    String, List[(String, Double)],
    String, List[(String, Double)]]
  (
```

```
    {f=>List(mapObtenirNoRefs(f))},
    {f=>f},
    {(f:String, s:List[(String, Double)])=>scala.List((f,s))},
    numWorkers,numWorkers,resultatComparacions
  )))

  //L'hi enviem el missatge de inicialització al MapReduce, després esperem el resultat,
      usant un pattern.
  val futur4 = act4 ? Iniciar()
  val resultatObtenirNoRefs =
      Await.result(futur4,timeout.duration).asInstanceOf[mutable.Map[String,
      List[(String, Double)]]].toList.sortBy(-_._2.length)
  //parem l'actor
  act4 ! PoisonPill
```

I per acabar el cas en què busquem parelles per sota de cert llindar que es referenciïn:

```
//donada llista referencies
//map: eliminar els que no arribin a cert llindar, despres eliminar els referenciats
def mapObtenirRefsDiferents(fitxer:(String, List[(String, Double)])):(String,
    List[(String, Double)]) = (
  fitxer._1,fitxer._2.filter(_._2>LlindarReferenciats).filter{
  f => diccionariFitxers(fitxer._1)._2.contains(f._1) ||
      diccionariFitxers(f._1)._2.contains(fitxer._1)
})
//Fem l'actor del MapReduce
val act5 = system.actorOf(Props (new MapReduceFramework[
  (String, List[(String, Double)]),
  String,List[(String, Double)],
  String, List[(String, Double)],
  String, List[(String, Double)]]
(
  {f=>List(mapObtenirRefsDiferents(f))},
  {f=>f},
  {(f:String, s:List[(String, Double)])=>scala.List((f,s))},
  numWorkers,numWorkers,resultatComparacions
)))

val futur5 = act5 ? Iniciar()
val resultatObtenirRefsDiferents =
    Await.result(futur5,timeout.duration).asInstanceOf[mutable.Map[String, List[(String,
    Double)]]].toList.sortBy(-_._2.length)
//parem l'actor
act5 ! PoisonPill
```

1.1.4.5 Volcatge dels resultats en un fitxer

Usant les funcions de entrada-sortida del Java, fiquem el contingut resultant en fitxers per a cada variable:

```
val output = "output/"+DirectoriFitxers + "/"
val nomDiccionari = output + "diccionari_"+n+".txt"
val nomVectorIDF = output + "vectorIDF_"+n+".txt"
val nomResultatComparacions = output + "resultatComparacions_"+n+".txt"
val nomResultatRefs1 = output + "resultatRefs1_"+n+".txt"
val nomResultatRefs2 = output + "resultatRefs2_"+n+".txt"

val fitxerInicial = new File(nomDiccionari)
fitxerInicial.getParentFile.mkdirs()
var pw = new PrintWriter(fitxerInicial)
diccionariFitxers.foreach{f=>
  pw.println("===" + f._1 + "\n====Llista Freqüències======")
  f._2._1.foreach{elem =>
    pw.println("   " + elem._1 + "-> " + elem._2.toString)
  }
  pw.println("====Llista Referències======")
  f._2._2.foreach{elem =>
    pw.println("   " + elem)
  }
  pw.print("\n\n")
}
pw.close()

pw = new PrintWriter(new File(nomVectorIDF))
diccionariIDF.foreach{f=>
  pw.println(f._1 + " -> " + f._2.toString)
}
pw.close()

pw = new PrintWriter(new File(nomResultatComparacions))
resultatComparacions.foreach{f=>
  if(f._2.nonEmpty) pw.println("===" + f._1 + "\n====Resultats Comparacions======")
  f._2.foreach{element=>
    pw.println("   " + element._1 + "-> " + element._2.toString)
  }
}
pw.close()

pw = new PrintWriter(new File(nomResultatRefs1))
resultatObtenirNoRefs.foreach{f=>
  if(f._2.nonEmpty) pw.println("===" + f._1 + "\n====Resultats Comparacions======")
  f._2.foreach{element=>
    pw.println("   " + element._1 + "-> " + element._2.toString)
  }
}
pw.close()

pw = new PrintWriter(new File(nomResultatRefs2))
resultatObtenirRefsDiferents.foreach{f=>
  if(f._2.nonEmpty) pw.println("===" + f._1 + "\n====Resultats Comparacions======")
  f._2.foreach{element=>
    pw.println("   " + element._1 + "-> " + element._2.toString)
  }
```

```
}
pw.close()
```

1.1.5 Main de l'aplicació

El main es divideix en dues parts:

- Una referent a la primera part de la pràctica.
- Una altre referent a la segona part de la pràctica

1.1.5.1 Primera part

```
override def main(args: Array[String]): Unit = {
  val filename = "pg11.txt"
  val fileContents = Source.fromFile(filename).mkString

  val filename2 = "english-stop.txt"
  val englishStopWords = getStopWords(filename2)

  val llistaFreq = freq(fileContents,1).sortBy(-_._2)
  val nParules = llistaFreq.foldLeft(0){(a,b) => b._2+a}
  val diff = llistaFreq.size
  println(String.format("%-20s %-10s %-20s %-20s", "Num de Parules:", nParules.toString,
      "Diferents", diff.toString))
  println(String.format("%-20s %-20s %-20s ", "Paraules", "ocurrencies", "frequencia"))
  println("------------------------------------------------------")
  val sFormat = "%-20s %-20s %-1.3f "
  for (p <- llistaFreq.slice(0,10)) println(String.format(sFormat, p._1, p._2.toString,
      (p._2*100.0/nParules).toFloat:java.lang.Float))

  println("\n\nCàlcul primer fitxer sense stopWords\n")
  println("debug")
  val llistaNonStop = nonStopFreq(fileContents, englishStopWords, 1).sortBy(-_._2)
  println(String.format("%-20s %-20s %-20s ", "Paraules", "ocurrencies", "frequencia"))
  println("------------------------------------------------------")
  for (p <- llistaNonStop.slice(0,10)) println(String.format(sFormat, p._1,
      p._2.toString, (p._2*100.0/nParules).toFloat:java.lang.Float))

  println("\n\nn-grames del primer fitxer amb n = 3\n")
  val llistaNgrames = freq(fileContents, 3).sortBy(-_._2)

  for (p <- llistaNgrames slice (0, 10)) println(String.format("%-30s %-5s", p._1,
      p._2.toString))

  println("Distribució de paraules")
  paraulafreqfreq(llistaFreq)

  val filename3 = "pg74.txt"
```

```
    println("\n\nComparació de pg11.txt amb pg74.txt utilitzant el cosinesim\n")

    val start = System.nanoTime()
    val newFileContents = Source.fromFile(filename).mkString
    val fileContents2 = Source.fromFile(filename3).mkString

    val freq1 = nonStopFreq(newFileContents, englishStopWords,1).sortBy(-_._2)
    val freq2 = nonStopFreq(fileContents2, englishStopWords, 1).sortBy(-_._2)

    val txt1 = freqAtf(freq1)
    val txt2 = freqAtf(freq2)

    val resCosinesim = cosinesim(txt1, txt2)

    val end = System.nanoTime()
    println(resCosinesim)
    println("El cosinesim ha tardat: " + (end-start).toDouble/1000000000.0)
...
  }
```

1.1.5.2 Segona Part

```
  println("\n\n\n\nINICI DEL CAMP MINAT: \n")

  val system = ActorSystem("SystemActor")

  val maxNgrames = 1

  for(n <- 1 to maxNgrames){
    calcularSimilituds(n,system)
  }

  system.terminate()
```

La funció de la segona part s'executa de 1 fins a maxNgrames, podent determinar quin es el màxim on volem arribar.

1.2 Fitxer MapReduceFramework.scala

Aquest fitxer inclou la classe amb el MapReduceFramework, que és la nostre implementació genèrica del mapReduce.

1.2.1 Idea General

La idea és un map reduce que rep unes dades, funció de mapping, funció de reducing, funció intermèdia, els nombres d'actors de mappeig i de reducció. Al rebre la crida Iniciar(), inicia el procediment. Usa la funció de mappeig sobre les dades, aplica la funció intermèdia amb els resultats, i sobre això aplica la funció de reducció.

1.2.2 Signatura

```
  class MapReduceFramework[Input, ClauMap, ValorMap, ClauIntermitja, ValorIntermig,
      ClauSortida, ValorSortida]
(
  mapFunction:Input=>List[(ClauMap, ValorMap)],
  funcioIntermitja:List[(ClauMap, ValorMap)] => List[(ClauIntermitja, ValorIntermig)] ,
  reduceFunction:(ClauIntermitja, ValorIntermig)=>List[(ClauSortida, ValorSortida)],
  nombreActorsMap: Int,
  nombreActorsReduce: Int,
  input: List[Input]
) extends Actor{
```

1.2.2.1 Sobre el Tipatge

- Input, el tipus de cada element de la Llista de dades entrada.
- ClauMap, el tipus de les claus resultants després del mappeig.
- ValorMap, el tipus dels valors resultants després del mappeig.
- ClauIntermitja, el tipus de les claus resultants després del tractament de dades intermedi.
- ValorIntermig, el tipus dels valors resultants després del tractament de dades intermedi.
- ClauSortida, el tipus de les claus resultants després de la funció de reducció.
- ValorSortida, el tipus dels valors resultants després de la funció de reducció.

1.2.3 Classes internes

```
  case class MissatgeMapeig(dades:Input)
  case class RespostaMapeig(dades:List[(ClauMap,ValorMap)])
  case class MissatgeReduccio(dades:(ClauIntermitja, ValorIntermig))
  case class RespostaReduccio(dades:List[(ClauSortida,ValorSortida)])
```

Les primeres case classes son per fer el *pattern matching* referent a la recepció de missatges.

1.2.4 Inicialització de valors

```
val inici: Long = System.nanoTime()
var pendent = 0
var pare:ActorRef = _
var llistaIntermedia:List[(ClauMap, ValorMap)] = Nil
var resultat:mutable.Map[ClauSortida,ValorSortida] =
    mutable.Map[ClauSortida,ValorSortida]()
```

S'inicialitza el temps de inici i els elements pendents de tractar. Creem variables pare (referent al creador del actor). LlistaIntermedia i resultat són la llista i el map (respectivament) a construir amb les dades que tornen el map i el reduce (respectivament).

1.2.5 Definició dels routers de Map i Reduce

```
val MapRouter: ActorRef = context.actorOf(RoundRobinPool(nombreActorsMap).props(Props(new
    Actor{
  override def receive: Receive = {
    case MissatgeMapeig(dades) => sender ! RespostaMapeig(mapFunction(dades))
  }})))
val ReduceRouter: ActorRef =
    context.actorOf(RoundRobinPool(nombreActorsReduce).props(Props( new Actor{
  override def receive: Receive = {
    case MissatgeReduccio(fitxer) => sender !
        RespostaReduccio(reduceFunction(fitxer._1,fitxer._2))
  }
}))
```

Per a la generació d'actors, hem utilitzat RoundRobin per crear un enroutador d'actors que s'encarregui del balanceig de càrregues i supervisió de fallades.

Els inicialitzem amb classes anònimes, que cada una rep un missatge ordenant mappeig o reducció i respon amb el resultat de aplicar a les dades la funció de de mappeig o reducció, segons el que toqui.

1.2.6 Funció de receive

```
override def receive: Receive = {
  case Iniciar() =>
    pare = sender
    println("Iniciem el MapReduceFramework")
    input.foreach{
      f=>MapRouter ! MissatgeMapeig(f)
        pendent+=1}
  case RespostaMapeig(dades) =>
    pendent-=1
    llistaIntermedia = dades ::: llistaIntermedia
```

```
    if (pendent == 0){
      context.stop(MapRouter)
      println("--------Duració mapeig: " +
          (System.nanoTime()-inici).toDouble/1000000000.0 + " segons")
      funcioIntermitja(llistaIntermedia).foreach {
        f => ReduceRouter ! MissatgeReduccio(f)
          pendent+=1
      }
      println("--------Duració Fins Intermig: " +
          (System.nanoTime()-inici).toDouble/1000000000.0 + " segons")
    }
   case RespostaReduccio(dades) =>
    pendent -=1
    dades.foreach{a => resultat+=(a._1->a._2)}
    if (pendent == 0){
      context.stop(ReduceRouter)
      val fi = System.nanoTime()
      println("Duració: " + (fi-inici).toDouble/1000000000.0 + " segons")
      pare ! resultat
    }
}
```

La funció es divideix en tres casos, en funció del missatge rebut:

1. Iniciar(): Aquest missatge és l'ordre de inici d'execució del mapReduce. Hem de guardar el valor del para, i enviem cada dada de la input al MapRouter, actualitzant el pendent
2. RespostaMapeig(): Missatge amb resposta d'un MapWorker, hem de actualitzar la llista de pendents, i afegir el rebut a la llista de valors intermedis, quan ja no en queden més, acabar el MapRouter, passar la funció intermèdia, i enviar cada dada d'aquesta al ReduceRouter.
3. RespostaReduccio(): Missatge amb resposta d'un ReduceWorker, hem de actualitzar la llista de pendents i actualitzar el resultat final. Quan no queden més, acabem el RouterWorker i responem al pare.

Capítol 2

Joc de proves

2.1 Resultats amb els fitxers donats

2.1.1 pg11.txt

```
Num de Parules:      30419      Diferents            3007
Paraules             ocurrencies          frequencia
------------------------------------------------------
the                  1818                 5.977
and                  940                  3.090
to                   809                  2.660
a                    690                  2.268
of                   631                  2.074
it                   610                  2.005
she                  553                  1.818
i                    545                  1.792
you                  481                  1.581
said                 462                  1.519


Càlcul primer fitxer sense stopWords

debug
Paraules             ocurrencies          frequencia
------------------------------------------------------
alice                403                  1.325
gutenberg            93                   0.306
project              87                   0.286
queen                75                   0.247
thought              74                   0.243
time                 71                   0.233
king                 63                   0.207
don                  61                   0.201
turtle               59                   0.194
began                58                   0.191
```

```
n-grames del primer fitxer amb n = 3

project gutenberg tm           57
the mock turtle                53
i don t                        31
the march hare                 30
said the king                  29
the project gutenberg          29
said the hatter                21
the white rabbit               21
said the mock                  19
said to herself                19
Distribució de paraules
Les 10 frequencies mes frequents:
1330 paraules apareixen 1 vegades
468 paraules apareixen 2 vegades
264 paraules apareixen 3 vegades
176 paraules apareixen 4 vegades
101 paraules apareixen 5 vegades
74 paraules apareixen 8 vegades
72 paraules apareixen 6 vegades
66 paraules apareixen 7 vegades
39 paraules apareixen 9 vegades
35 paraules apareixen 10 vegades
Les 5 frequencies menys frequents:
1 paraules apareixen 74 vegades
1 paraules apareixen 1818 vegades
1 paraules apareixen 610 vegades
1 paraules apareixen 200 vegades
1 paraules apareixen 358 vegades
```

2.1.2 pg11-net.txt

```
Num de Parules:      27341      Diferents            2572
Paraules             ocurrencies          frequencia
-------------------------------------------------------
the                  1644                 6.013
and                  872                  3.189
to                   729                  2.666
a                    632                  2.312
it                   595                  2.176
she                  553                  2.023
i                    545                  1.993
of                   514                  1.880
said                 462                  1.690
you                  411                  1.503


Càlcul primer fitxer sense stopWords

debug
Paraules             ocurrencies          frequencia
```

```
-----------------------------------------------------
alice                398                  1.456
queen                75                   0.274
thought              74                   0.271
time                 71                   0.260
king                 63                   0.230
don                  61                   0.223
turtle               59                   0.216
began                58                   0.212
ll                   57                   0.208
mock                 56                   0.205


n-grames del primer fitxer amb n = 3

the mock turtle               53
i don t                       31
the march hare                30
said the king                 29
said the hatter               21
the white rabbit              21
said the mock                 19
said to herself               19
said the caterpillar          18
she said to                   17
Distribució de paraules
Les 10 frequencies mes frequents:
1116 paraules apareixen 1 vegades
396 paraules apareixen 2 vegades
229 paraules apareixen 3 vegades
144 paraules apareixen 4 vegades
91 paraules apareixen 5 vegades
62 paraules apareixen 7 vegades
61 paraules apareixen 6 vegades
55 paraules apareixen 8 vegades
36 paraules apareixen 10 vegades
35 paraules apareixen 9 vegades
Les 5 frequencies menys frequents:
1 paraules apareixen 125 vegades
1 paraules apareixen 218 vegades
1 paraules apareixen 369 vegades
1 paraules apareixen 74 vegades
1 paraules apareixen 632 vegades
```

2.1.3 pg12.txt

```
Num de Parules:      33710      Diferents            3192
Paraules             ocurrencies          frequencia
-----------------------------------------------------
the                  1775                 5.266
and                  975                  2.892
a                    819                  2.430
to                   817                  2.424
```

```
you                 686                 2.035
it                  681                 2.020
i                   660                 1.958
of                  604                 1.792
she                 544                 1.614
said                473                 1.403


Càlcul primer fitxer sense stopWords

debug
Paraules            ocurrencies         frequencia
-----------------------------------------------------
alice               455                 1.350
queen               185                 0.549
gutenberg           93                  0.276
ll                  89                  0.264
project             87                  0.258
thought             86                  0.255
don                 80                  0.237
time                70                  0.208
red                 69                  0.205
white               67                  0.199


n-grames del primer fitxer amb n = 3

project gutenberg tm          57
the red queen                 54
i don t                       44
the white queen               33
the project gutenberg         31
said in a                     21
she went on                   18
gutenberg tm electronic       18
said the red                  17
the queen said                16
Distribució de paraules
Les 10 frequencies mes frequents:
1367 paraules apareixen 1 vegades
501 paraules apareixen 2 vegades
300 paraules apareixen 3 vegades
186 paraules apareixen 4 vegades
138 paraules apareixen 5 vegades
97 paraules apareixen 6 vegades
66 paraules apareixen 7 vegades
57 paraules apareixen 8 vegades
37 paraules apareixen 9 vegades
35 paraules apareixen 12 vegades
Les 5 frequencies menys frequents:
1 paraules apareixen 131 vegades
1 paraules apareixen 975 vegades
1 paraules apareixen 136 vegades
1 paraules apareixen 157 vegades
```

```
1 paraules apareixen 358 vegades
```

2.1.4 pg12-net.txt

```
Num de Parules:      30624      Diferents            2748
Paraules             ocurrencies          frequencia
------------------------------------------------------
the                  1594                 5.205
and                  907                  2.962
a                    761                  2.485
to                   737                  2.407
it                   666                  2.175
i                    660                  2.155
you                  616                  2.011
she                  544                  1.776
of                   485                  1.584
said                 473                  1.545


Càlcul primer fitxer sense stopWords

debug
Paraules             ocurrencies          frequencia
------------------------------------------------------
alice                455                  1.486
queen                185                  0.604
ll                   89                   0.291
thought              86                   0.281
don                  80                   0.261
time                 70                   0.229
red                  69                   0.225
white                67                   0.219
king                 66                   0.216
head                 63                   0.206


n-grames del primer fitxer amb n = 3

the red queen                  54
i don t                        44
the white queen                33
said in a                      21
she went on                    18
said the red                   17
the queen said                 16
thought to herself             16
don t know                     15
the knight said                14
Distribució de paraules
Les 10 frequencies mes frequents:
1128 paraules apareixen 1 vegades
452 paraules apareixen 2 vegades
258 paraules apareixen 3 vegades
```

```
156 paraules apareixen 4 vegades
129 paraules apareixen 5 vegades
79 paraules apareixen 6 vegades
60 paraules apareixen 7 vegades
47 paraules apareixen 8 vegades
37 paraules apareixen 12 vegades
30 paraules apareixen 9 vegades
Les 5 frequencies menys frequents:
1 paraules apareixen 323 vegades
1 paraules apareixen 74 vegades
1 paraules apareixen 52 vegades
1 paraules apareixen 1594 vegades
1 paraules apareixen 157 vegades
```

2.1.5 pg74.txt

```
Num de Parules:      77488      Diferents            7625
Paraules             ocurrencies          frequencia
-------------------------------------------------------
the                  3973                 5.127
and                  3193                 4.121
a                    1955                 2.523
to                   1807                 2.332
of                   1585                 2.045
it                   1332                 1.719
he                   1256                 1.621
was                  1170                 1.510
that                 1044                 1.347
i                    1018                 1.314


Càlcul primer fitxer sense stopWords

debug
Paraules             ocurrencies          frequencia
-------------------------------------------------------
tom                  824                  1.063
huck                 258                  0.333
ll                   232                  0.299
don                  224                  0.289
time                 191                  0.246
joe                  170                  0.219
boys                 158                  0.204
boy                  136                  0.176
ain                  123                  0.159
back                 121                  0.156


n-grames del primer fitxer amb n = 3

i don t                        70
project gutenberg tm           57
there was a                    44
```

```
don t you                       35
by and by                       32
the project gutenberg           31
don t know                      25
there was no                    25
it ain t                        24
i won t                         22
Distribució de paraules
Les 10 frequencies mes frequents:
3640 paraules apareixen 1 vegades
1269 paraules apareixen 2 vegades
642 paraules apareixen 3 vegades
423 paraules apareixen 4 vegades
240 paraules apareixen 5 vegades
189 paraules apareixen 6 vegades
169 paraules apareixen 7 vegades
134 paraules apareixen 8 vegades
96 paraules apareixen 10 vegades
85 paraules apareixen 9 vegades
Les 5 frequencies menys frequents:
1 paraules apareixen 232 vegades
1 paraules apareixen 131 vegades
1 paraules apareixen 301 vegades
1 paraules apareixen 157 vegades
1 paraules apareixen 146 vegades
```

2.1.6 pg74-net.txt

```
Num de Parules:      74388      Diferents            7303
Paraules             ocurrencies          frequencia
-----------------------------------------------------
the                  3794                 5.100
and                  3125                 4.201
a                    1897                 2.550
to                   1727                 2.322
of                   1463                 1.967
it                   1317                 1.770
he                   1253                 1.684
was                  1168                 1.570
that                 1029                 1.383
i                    1018                 1.369


Càlcul primer fitxer sense stopWords

debug
Paraules             ocurrencies          frequencia
-----------------------------------------------------
tom                  819                  1.101
huck                 258                  0.347
ll                   232                  0.312
don                  224                  0.301
time                 191                  0.257
```

```
joe                 170                 0.229
boys                158                 0.212
boy                 136                 0.183
ain                 123                 0.165
back                121                 0.163


n-grames del primer fitxer amb n = 3

i don t                         70
there was a                     44
don t you                       35
by and by                       32
don t know                      25
there was no                    25
it ain t                        24
i won t                         22
out of the                      22
i can t                         21
Distribució de paraules
Les 10 frequencies mes frequents:
3530 paraules apareixen 1 vegades
1211 paraules apareixen 2 vegades
612 paraules apareixen 3 vegades
392 paraules apareixen 4 vegades
222 paraules apareixen 5 vegades
169 paraules apareixen 6 vegades
163 paraules apareixen 7 vegades
126 paraules apareixen 8 vegades
92 paraules apareixen 10 vegades
82 paraules apareixen 9 vegades
Les 5 frequencies menys frequents:
1 paraules apareixen 1253 vegades
1 paraules apareixen 893 vegades
1 paraules apareixen 52 vegades
1 paraules apareixen 301 vegades
1 paraules apareixen 146 vegades
```

2.1.7 pg2500.txt

```
Num de Parules:      42864      Diferents            3955
Paraules             ocurrencies          frequencia
--------------------------------------------------------
the                  2221                 5.182
and                  1434                 3.345
to                   1225                 2.858
of                   1106                 2.580
a                    969                  2.261
he                   960                  2.240
his                  708                  1.652
in                   686                  1.600
you                  540                  1.260
had                  524                  1.222
```

```
Càlcul primer fitxer sense stopWords

debug
Paraules             ocurrencies          frequencia
------------------------------------------------------
siddhartha           409                  0.954
govinda              146                  0.341
time                 139                  0.324
river                110                  0.257
long                 97                   0.226
gutenberg            93                   0.217
man                  90                   0.210
life                 88                   0.205
project              87                   0.203
kamala               83                   0.194


n-grames del primer fitxer amb n = 3

a long time                    59
project gutenberg tm           57
for a long                     47
the project gutenberg          31
the exalted one                24
he had been                    19
in his heart                   19
gutenberg tm electronic        18
in order to                    16
with a smile                   16
Distribució de paraules
Les 10 frequencies mes frequents:
1732 paraules apareixen 1 vegades
667 paraules apareixen 2 vegades
330 paraules apareixen 3 vegades
230 paraules apareixen 4 vegades
127 paraules apareixen 5 vegades
116 paraules apareixen 6 vegades
84 paraules apareixen 7 vegades
61 paraules apareixen 8 vegades
48 paraules apareixen 9 vegades
44 paraules apareixen 10 vegades
Les 5 frequencies menys frequents:
1 paraules apareixen 1434 vegades
1 paraules apareixen 52 vegades
1 paraules apareixen 960 vegades
1 paraules apareixen 131 vegades
1 paraules apareixen 146 vegades
```

2.1.8 pg2500-net.txt

```
Num de Parules:      39774      Diferents            3551
```

```
Paraules             ocurrencies          frequencia
-----------------------------------------------------
the                  2045                 5.142
and                  1365                 3.432
to                   1145                 2.879
of                   987                  2.482
he                   957                  2.406
a                    911                  2.290
his                  708                  1.780
in                   629                  1.581
had                  524                  1.317
was                  511                  1.285


Càlcul primer fitxer sense stopWords

debug
Paraules             ocurrencies          frequencia
-----------------------------------------------------
siddhartha           404                  1.016
govinda              146                  0.367
time                 139                  0.349
river                110                  0.277
long                 96                   0.241
man                  90                   0.226
life                 87                   0.219
kamala               83                   0.209
thought              82                   0.206
love                 82                   0.206


n-grames del primer fitxer amb n = 3

a long time                    59
for a long                     47
the exalted one                24
he had been                    19
in his heart                   19
in order to                    16
with a smile                   16
in the forest                  15
which he had                   15
he did not                     14
Distribució de paraules
Les 10 frequencies mes frequents:
1558 paraules apareixen 1 vegades
587 paraules apareixen 2 vegades
295 paraules apareixen 3 vegades
195 paraules apareixen 4 vegades
116 paraules apareixen 5 vegades
105 paraules apareixen 6 vegades
78 paraules apareixen 7 vegades
51 paraules apareixen 8 vegades
43 paraules apareixen 9 vegades
```

```
41 paraules apareixen 10 vegades
Les 5 frequencies menys frequents:
1 paraules apareixen 254 vegades
1 paraules apareixen 91 vegades
1 paraules apareixen 316 vegades
1 paraules apareixen 629 vegades
1 paraules apareixen 146 vegades
```

2.2 Resultat amb Fitxer personalitzat

2.2.1 Contingut del fitxer

```
Hola, Adéu
Estic fent la Pràctica de Programació Declarativa
Hola, Adéu
Estic fent la Pràctica de Programació Declarativa
Hola, Adèu
Estic fent la Pràctica de Programació Declarativa
Ja he fet repeticions.
```

2.2.2 Resultat

```
Num de Parules:      31         Diferents            14
Paraules             ocurrencies          frequencia
-------------------------------------------------------
declarativa          3                    9.677
la                   3                    9.677
de                   3                    9.677
fent                 3                    9.677
programació          3                    9.677
pràctica             3                    9.677
estic                3                    9.677
hola                 3                    9.677
adéu                 2                    6.452
fet                  1                    3.226


Càlcul primer fitxer sense stopWords

debug
Paraules             ocurrencies          frequencia
-------------------------------------------------------
declarativa          3                    9.677
fent                 3                    9.677
programació          3                    9.677
pràctica             3                    9.677
hola                 3                    9.677
fet                  1                    3.226
adèu                 1                    3.226
repeticions          1                    3.226
```

```
n-grames del primer fitxer amb n = 3

fent la pràctica              3
la pràctica de                3
de programació declarativa    3
pràctica de programació       3
estic fent la                 3
hola adéu estic               2
programació declarativa hola  2
adéu estic fent               2
declarativa hola adéu         1
declarativa hola adèu         1
Distribució de paraules
Les 10 frequencies mes frequents:
8 paraules apareixen 3 vegades
5 paraules apareixen 1 vegades
1 paraules apareixen 2 vegades
Les 5 frequencies menys frequents:
8 paraules apareixen 3 vegades
5 paraules apareixen 1 vegades
1 paraules apareixen 2 vegades
```

2.3 Resultats Cosinesim

Les proves d'aquesta secció han estat fetes amb la següent màquina:

Les següents proves s'han fet amb la següent màquina:

- CPU: AMD Ryzen 3 1200 amb 4 cores i 4 threads a una freqüència base de 3.10GHz i una freqüència turbo 3.4GHz
- RAM: 8GB
- Almacenatge: HDD

2.3.1 pg-11.txt amb pg11-net.txt

Resultat: 0.9532862678863194

El cosinesim ha tardat: 0.376971377 segons

2.3.2 pg-11.txt amb pg74.txt

Resultat: 0.29079706998634325

El cosinesim ha tardat: 0.460658575 segons

2.3.3 pg-12.txt amb pg74.txt

Resultat: 0.2898070543611612

El cosinesim ha tardat: 0.463819116 segons

2.3.4 pg-12.txt amb pg-12.txt

Resultat: 0.9999999999997916

El cosinesim ha tardat: 0.416221798 segons

Capítol 3

Resultats

El temps d'execució al calcular l'exercici amb la maquina esmentada aqui baix és d'aproximadament 55 minuts. Per això, hem creat dos subsets de test per poder provar canvis al codi fàcilment, un que conté tots els arxius acabats en "91.xml", i un altre amb tots els arxius acabats en "9.xml". Els resultats de la nostra execució general estan penjats a `https://drive.google.com/file/d/1LoqFrBAbluANe4F8Xv41rmG6mQjydSAV/view?usp=sharing`

3.1 Taula de rendiment de MapReduce segons nombre d'actors

Les següents proves s'han fet amb la següent màquina:

- CPU: Intel Core i5 6600 amb 4 cores i 4 threads a una freqüència base de 3.30GHz i una freqüència turbo de 3.90GHz
- RAM: 16GB
- Emmagatzematge: Samsung 960 EVO NVMe

Nombre d'actors	Temps transcorregut
1	150.21s
2	76.03s
4	47.00s
8	44.33s
16	45.48s
32	48.55s
64	44.35s
128	46.25

3.2 Taula d'articles a referenciar segons llindar

Hem decidit usar un llindar de 0.01 basant-nos en els resultats obtinguts en tests més petits. La taula obtinguda en aquests tests és la següent:

Llindar	Nombre d'articles a referenciar
0.001	97100
0.005	59440
0.01	30119
0.05	2209
0.1	567
0.5	7

3.3 Ex.3

Sembla que canviar la longitud dels n-grames de 1 a 5 no afecta al nombre de pagines similars que no es referencien. Suposem que això és degut a que el idf canviaria proporcionalment al tf, i els canvis als dos, al fer les comparacions, es cancelarien i ens quedaria el mateix resultat.