

# Pràctica de programació funcional + orientada a objectes

## Similitud entre documents

Ismael El Habri, Lluís Trilla

16 d'octubre de 2018

# Índex

<b>1</b>	<b>Codi de la pràctica</b>	<b>4</b>
1.1	Fitxer SimilitudEntreDocuments.scala . . . . .	4
1.1.1	Funcions de freqüència . . . . .	4
1.1.1.1	Funció Freqüència . . . . .	4
1.1.1.2	Funció de normalització . . . . .	5
1.1.1.3	Funció de Freqüències sense <i>StopWords</i> . . . . .	5
1.1.1.4	Funció de distribució de paraules . . . . .	5
1.1.2	Funcions de Comparació . . . . .	6
1.1.2.1	Funcions auxiliars . . . . .	6
1.1.2.2	Funció cosinesim . . . . .	6
1.1.3	Funcions pel MapReduce . . . . .	7
1.1.3.1	Funció per Llegir fitxers XML . . . . .	7
1.1.3.2	Llista de fitxers en un directori . . . . .	7
1.1.4	Funció del Apartat 2 de la Pràctica . . . . .	7
1.1.4.1	Llegir Fitxers i tractar-los . . . . .	8
1.1.4.2	Càlcul del vector IDF . . . . .	9
1.1.4.3	Comparacions tots amb tots . . . . .	10
1.1.4.4	Tractant referències . . . . .	11
1.1.4.5	Volcatge dels resultats en un fitxer . . . . .	12
1.1.5	Main de l'aplicació . . . . .	14
1.1.5.1	Primera part . . . . .	14

1.1.5.2	Segona Part . . . . .	15
1.2	Fitxer MapReduceFramework.scala . . . . .	15
1.2.1	Idea General . . . . .	15
1.2.2	Signatura . . . . .	16
1.2.2.1	Sobre el Tipatge . . . . .	16
<b>2</b>	<b>Joc de proves</b>	<b>17</b>
<b>3</b>	<b>Resultats</b>	<b>18</b>

# 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 *Stop Words*

---

```
//Rep una String i una llista de Strings amb stop words, i fa el vector de freqüències
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 freqüències més freqüents, i les 5 menys freqüents
def paraulaFreqFreq(llistaFreqüencies:List[(String,Int)]): Unit = {
  val stringFreqüencies:String = llistaFreqüencies.map(_._2.toString.concat(" ")).mkString
  val freqFreqList = stringFreqüencies
    .split(" ").groupBy(identity).mapValues(_.length).toList.sortBy(_._2)
  println("Les 10 freqüencies mes freqüents:")
  for(frequencia <- freqFreqList.slice(0,10))
    println(frequencia._2 + " paraules apareixen " + frequencia._1 + " vegades")
  println("Les 5 freqüencies menys freqüents:")
  for(frequencia <-
    freqFreqList.slice(freqFreqList.length-5,freqFreqList.length).sortBy(_._2))
    println(frequencia._2 + " paraules apareixen " + frequencia._1 + " vegades")
}
```

---

### 1.1.2 Funcions de Comparació

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

#### 1.1.2.1 Funcions auxiliars

---

```
//Rep dos vectors de freqüencies 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 freqüencia = 0 a txt1
// Al final ordenem alfabeticament, per tenir el mateix ordre en els dos vectors!
def alinearResult(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 dona l'element més freqüent de la llista de freqüències.
3. *alinearResult*: Donat un vector *a* alinear, i un vector amb el qual s'ha de alinear, alinea el vector *a* 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) <- alinearResult(txt1,txt2) zip alinearResult(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 similitud:

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

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 freqüències.

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 document 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 referències internes (les que contenen el caràcter '#'), i treure la part de les referències 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, i:

---

```
def calcularSimilituds(n: Int, system: ActorSystem)= {
  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, Referències), calcula les
//freqüències 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)))},
    10,10,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)))},
    10,10,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 multiplica 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)))},
  100,100,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 referencïin i fitxers que es referencïin 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 referencïin.
- 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 referencïin:

---

```

//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))},
    100, 100, 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))},
    100,100,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, fem 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", "ocurrences", "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", "ocurrences", "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\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

Cal definir-li el tipus d'entrada



## Capítol 2

### Joc de proves

## Capítol 3

## Resultats