Informe de l'Entrega Funcional + Objectes 1ra part

Guillem Vidal

1 Coses a tenir en compte

He modificat les signatures de la majoria de funcions per tal que acceptin un canvi de funcionament. No m'agradava el fet de tenir en memòria el fitxer sencer a l'hora de tractar-lo, ja que podriem estar llegint fitxers molt grans i llavors el programa no seria realment escalable. A més que no cal tenir-los sencers en memòria, els podem llegir en blocs.

Aquesta solució no té perquè alentir massa el programa, no crec que ni tan sols es noti.

El programa assumeix que els fitxers de proves estan localitzats a la carpeta `input`, relativa al *Working Directory*.

2 Funcions

Utilitzo una funció per separar el contingut en paraules, filtrar per paraules que siguin alfanumèriques, i passar-les a minúscules:

```
def words(contents: String): View[String] =
    contents.view
        .map(char => if (char.isLetterOrDigit) { char } else { ' ' })
            // Substituim per espais els caracters que no son
            alfanumerics
        .mkString // Construim una string de la cadena de caracters
            resultant
        .split("\\s").view // Dividim la string per espais
        .filter(str => str.nonEmpty) // Esborrem de la iteracio les
            strings buides
        .map(str => str.toLowerCase) // Passem les strings que queden
            a minuscules
```

2.1 Freqüències de paraula

Per computar les freqüències de paraula utilitzo:

```
    def freq(sentences: Iterable[String]): Map[String, Int] =
        sentences.groupMapReduce(identity)(_ => 1)(_ + _) // Agrupem les
            frases resultants en un mapa d'aparicions
```

Després les puc escriure per pantalla mitjançant:

```
    def showFrequencies(list: List[(String, Int)]): Unit = {

        val freqs = list.sortBy(el => el._2)
        val total = freqs.map(el => el._2).sum
        val diff = freqs.size

        println(f"Num de Paraules:\t$total\tDiferents:\t$diff")
        println("Paraules\tOccurrencies\tFrequencia")
        println("-------------------------------")

        freqs.view
            .drop(freqs.size - 10)
            .toList
            .reverse
            .foreach(el => {
                val word = el._1
                val count = el._2
                val freq = count.toFloat / total * 100
                println(f"$word\t$count\t$freq")
            })

    }
```

Llavors, si executo:

```
        val lines = fromFile("input" + File.separator +
            "pg11.txt").getLines().to(Iterable)
        val freqs = freq(lines.flatMap(words))
        showFrequencies(freqs.toList)
```

Escriu per consola:

```
Num de Paraules: 30530 Diferents: 3041
Paraules Ocurrències Freqüència
-------------------------------
the 1818 5.9547987
and 940 3.0789387
to 809 2.6498528
a 690 2.260072
of 631 2.0668197
it 610 1.9980347
she 553 1.8113332
```

```
i 545 1.7851293
you 481 1.5754995
said 462 1.5132656
```

2.2 Sense stop-words

Definim la funció de manera que:

```
    val stopwords = fromFile("input/english-stop.txt").getLines().toSet;

   def nonstopwords(words: Iterable[String], stopwords: Set[String]):
       Iterable[String] =
       words.filter(word => !stopwords.contains(word)) // Filtrem les
           paraules que siguin stopwords

    def nonstopfreq(words: Iterable[String], stopwords: Set[String]):
        Map[String, Int] =
       freq(nonstopwords(words, stopwords))
```

I executant:

```
       val lines = fromFile("input" + File.separator +
           "pg11.txt").getLines().to(Iterable)
       val freqs = nonstopfreq(lines.flatMap(words), stopwords)
       showFrequencies(freqs.toList)
```

Obtenim:

```
Num de Paraules: 10150 Diferents: 2657
Paraules Ocurrències Freqüència
-------------------------------
alice 403 3.9704435
gutenberg 93 0.9162562
project 87 0.8571428
queen 75 0.7389163
thought 74 0.72906405
time 71 0.6995074
king 63 0.6206897
turtle 59 0.58128077
began 58 0.57142854
tm 57 0.56157637
```

2.3 Distribució de paraules

Definim la funció tal que:

```
    def paraulafreqfreq(lines: Iterable[String]): Unit = {
```

```
    val list = nonstopfreq(lines.flatMap(words), stopwords)
    val freqsFreq = list.values
        .groupMapReduce(identity)(_ => 1)(_ + _)
        .to(SortedMap)

    println("Les 10 frequencies mes frequents:")
    val top = freqsFreq.view.take(10)
    for ((k, v) <- top) {
        println(f"$v paraules apareixen $k vegades")
    }

    println("Les 5 frequencies menys frequents:")
    val bot = freqsFreq.view.drop(freqsFreq.size - 5).toList.reverse
    for ((k, v) <- bot) {
        println(f"$v paraules apareixen $k vegades")
    }

}
```

Si executem:

```
    val lines = fromFile("input" + File.separator +
        "pg11.txt").getLines().to(Iterable)
    paraulafreqfreq(lines.flatMap(words))
```

Obtenim:

```
Les 10 freqüències més freqüents:
1309 paraules apareixen 1 vegades
443 paraules apareixen 2 vegades
248 paraules apareixen 3 vegades
166 paraules apareixen 4 vegades
84 paraules apareixen 5 vegades
58 paraules apareixen 6 vegades
60 paraules apareixen 7 vegades
60 paraules apareixen 8 vegades
32 paraules apareixen 9 vegades
28 paraules apareixen 10 vegades
Les 5 freqüències menys freqüents:
1 paraules apareixen 403 vegades
1 paraules apareixen 93 vegades
1 paraules apareixen 87 vegades
1 paraules apareixen 75 vegades
1 paraules apareixen 74 vegades
```

2.4 ngrames

Definim la funció de manera que:

```
def ngrames(lines: Iterable[String], wordcount: Int):
    Iterable[String] =
   nonstopwords(lines.flatMap(words), stopwords) // Filtrem per
       stopwords
      .sliding(wordcount) // [0, 1, 2, 3].sliding(2) = [[0, 1], [1,
         2], [2, 3]]
      .map(iter => iter.mkString(" ")) // Formem una string
         separada per espais
      .to(Iterable)
```

I si executem:

```
val lines = fromFile("input" + File.separator +
    "pg11.txt").getLines().to(Iterable)
val freqs = ngrames(lines, 3)
showFrequencies(freqs.toList)
```

Obtenim:

```
Num de Paraules: 10148 Diferents: 9721
Paraules Ocurrències Freqüència
-------------------------------
project gutenberg tm 57 0.56168705
gutenberg tm electronic 18 0.17737485
gutenberg literary archive 13 0.12810406
literary archive foundation 13 0.12810406
project gutenberg literary 13 0.12810406
tm electronic works 12 0.1182499
gutenberg tm license 8 0.07883327
full project gutenberg 7 0.068979114
alice adventures wonderland 6 0.05912495
join dance join 6 0.05912495
```

2.5 Vector space model

Definim la funció tal que:

```
def cosinesim(fst: Iterable[String], sec: Iterable[String],
    wordcount: Int): Float = {

   val fstFreq = freq(ngrames(fst, wordcount))
   val secFreq = freq(ngrames(sec, wordcount))

   val fstMax = fstFreq.values.max
   val secMax = secFreq.values.max

   val fstExtended = fstFreq.view.concat(
            secFreq.keys.flatMap(word => fstFreq.get(word) match {
                case Some(_) => None
                case None => Some((word, 0))
            })
        ).toList

        val paired = fstExtended.map(el => el._2).zip(fstExtended.map(el
            => secFreq.get(el._1) match {
            case Some(freq) => freq
            case None => 0
        }))

        var dot = 0f
        var aSqrLen = 0f
        var bSqrLen = 0f

        paired.foreach(pair => {
            val (f, s) = pair
            val (a, b) = (f.toFloat / fstMax, s.toFloat / secMax)

            dot += a * b
            aSqrLen += a * a
            bSqrLen += b * b
        })

        dot / (Math.sqrt(aSqrLen) * Math.sqrt(bSqrLen)).toFloat
    }

    def simil(fst_name: String, sec_name: String): Float = {

        val fst = fromFile(fst_name).getLines().to(Iterable)
        val sec = fromFile(sec_name).getLines().to(Iterable)

        cosinesim(fst, sec, 1)
    }
```

I si executem:

```
        println(simil("input" + File.separator + "pg11.txt", "input" +
            File.separator + "pg12.txt"))
```

Obtenim:

```
0.8771395
```

3 Altres proves

3.1 Similituds mitjançant digrames i trigrames

Executem:

```
val fst_name = "input" + File.separator + "pg11.txt"
val sec_name = "input" + File.separator + "pg12.txt"

val fst = fromFile(fst_name).getLines().to(Iterable)
val sec = fromFile(sec_name).getLines().to(Iterable)

val one = cosinesim(fst, sec, 1)
val two = cosinesim(fst, sec, 2)
val three = cosinesim(fst, sec, 3)

println("One: " + one)
println("Two: " + two)
println("Three: " + three)
```

I obtenim:

```
One: 0.8771395
Two:0.5082349
Three: 0.39970124
```

3.2 Comparació entre altres documents

Executem:

```
val files = Array("pg11", "pg11-net", "pg12", "pg12-net",
    "pg74", "pg74-net", "pg2500", "pg2500-net")
for (fst <- files) {
    for (sec <- files) {
        val fst_name = "input" + File.separator + fst + ".txt"
        val sec_name = "input" + File.separator + sec + ".txt"

        val result = simil(fst_name, sec_name)

        println("Comparison between " + fst + " and " + sec + ":
            " + result)
    }
}
```

I obtenim:

```
Comparison between pg11 and pg11: 1.0
Comparison between pg11 and pg11-net: 0.9481747
Comparison between pg11 and pg12: 0.8771395
Comparison between pg11 and pg12-net: 0.8209936
Comparison between pg11 and pg74: 0.26102832
Comparison between pg11 and pg74-net: 0.21418925
Comparison between pg11 and pg2500: 0.28081235
Comparison between pg11 and pg2500-net: 0.20754851
Comparison between pg11-net and pg11: 0.9481736
Comparison between pg11-net and pg11-net: 1.0
Comparison between pg11-net and pg12: 0.82971895
Comparison between pg11-net and pg12-net: 0.8641726
Comparison between pg11-net and pg74: 0.21763502
Comparison between pg11-net and pg74-net: 0.21931946
Comparison between pg11-net and pg2500: 0.20846006
Comparison between pg11-net and pg2500-net: 0.21318485
Comparison between pg12 and pg11: 0.8771409
Comparison between pg12 and pg11-net: 0.8297241
Comparison between pg12 and pg12: 1.0
Comparison between pg12 and pg12-net: 0.9598648
Comparison between pg12 and pg74: 0.24946001
Comparison between pg12 and pg74-net: 0.20808594
Comparison between pg12 and pg2500: 0.2648699
Comparison between pg12 and pg2500-net: 0.20018151
Comparison between pg12-net and pg11: 0.82099396
Comparison between pg12-net and pg11-net: 0.86417675
Comparison between pg12-net and pg12: 0.95986295
Comparison between pg12-net and pg12-net: 1.0
Comparison between pg12-net and pg74: 0.20836323
Comparison between pg12-net and pg74-net: 0.20976494
Comparison between pg12-net and pg2500: 0.1982276
Comparison between pg12-net and pg2500-net: 0.20229337
Comparison between pg74 and pg11: 0.26102868
Comparison between pg74 and pg11-net: 0.21763316
Comparison between pg74 and pg12: 0.24945965
Comparison between pg74 and pg12-net: 0.20836194
Comparison between pg74 and pg74: 1.0
Comparison between pg74 and pg74-net: 0.98813325
Comparison between pg74 and pg2500: 0.30113128
Comparison between pg74 and pg2500-net: 0.26767817
Comparison between pg74-net and pg11: 0.2141945
Comparison between pg74-net and pg11-net: 0.21932364
Comparison between pg74-net and pg12: 0.20809072
Comparison between pg74-net and pg12-net: 0.20976995
Comparison between pg74-net and pg74: 0.9881346
Comparison between pg74-net and pg74-net: 1.0
Comparison between pg74-net and pg2500: 0.26487228
Comparison between pg74-net and pg2500-net: 0.26897246
Comparison between pg2500 and pg11: 0.2808145
Comparison between pg2500 and pg11-net: 0.2084602
Comparison between pg2500 and pg12: 0.26487142
Comparison between pg2500 and pg12-net: 0.19822833
Comparison between pg2500 and pg74: 0.30113342
Comparison between pg2500 and pg74-net: 0.26486856
Comparison between pg2500 and pg2500: 1.0
Comparison between pg2500 and pg2500-net: 0.96923983
Comparison between pg2500-net and pg11: 0.20754899
Comparison between pg2500-net and pg11-net: 0.21318404
Comparison between pg2500-net and pg12: 0.20018135
Comparison between pg2500-net and pg12-net: 0.20229314
Comparison between pg2500-net and pg74: 0.26767913
Comparison between pg2500-net and pg74-net: 0.26896688
Comparison between pg2500-net and pg2500: 0.96924025
Comparison between pg2500-net and pg2500-net: 1.0
```