Sergi López i Pablo Pozo

Part 1

1. Càlcul de freqüències de paraules (freq)

- **Funció freq**: Processa el text per calcular la freqüència de cada paraula. El text es normalitza a minúscules i es neteja de caràcters especials. Després es divideix en paraules, es comptabilitzen les ocurrències de cada paraula, i es retorna una llista ordenada descendentment per freqüència.
- **Lectura del text**: El fitxer pg11.txt es llegeix, convertint el contingut en un sol string. El fitxer es tanca després de llegir-lo.
- **Càlcul i impressió**:
    - Es calcula el nombre total de paraules i les paraules úniques.
    - Es mostren les 10 paraules més freqüents amb la seva freqüència relativa (percentatge respecte al total de paraules).

```
Num de paraules: 30419    Diferents: 3007
Paraules Ocurrències Freqüència
------------------------------
the: 1818 5,98
and: 940 3,09
to: 809 2,66
a: 690 2,27
of: 631 2,07
it: 610 2,01
she: 553 1,82
i: 545 1,79
you: 481 1,58
said: 462 1,52
```

---

2. Càlcul de freqüències sense paraules buides (nonstopfreq)

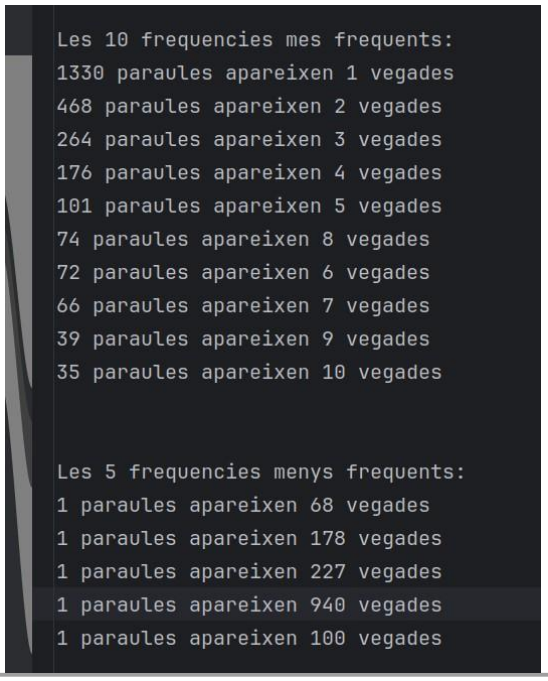
- **Funció nonstopfreq**: Similar a freq, però elimina paraules buides (stop words) definides en un conjunt de paraules que no tenen significat rellevant, com "el", "de", "i".
- **Lectura de les paraules buides**: Llegeix les paraules buides del fitxer english-stop.txt i les guarda en un Set.
- **Processament sense paraules buides**:
    - Torna a calcular les freqüències de les paraules sense les paraules buides.
    - Es mostra la informació estadística i les 10 paraules més freqüents sense paraules buides.

Sergi López i Pablo Pozo

```
Num de paraules: 10038    Diferents: 2623
Paraules Ocurrències Freqüència
------------------------------
alice: 403 4,01
gutenberg: 93 0,93
project: 87 0,87
queen: 75 0,75
thought: 74 0,74
time: 71 0,71
king: 63 0,63
turtle: 59 0,59
began: 58 0,58
tm: 57 0,57
```

3. Distribució de freqüències de paraules (paraulafreqfreq)

- **Funció paraulafreqfreq**:
    - Utilitza freq per obtenir les freqüències de les paraules.
    - Calcula quantes paraules tenen la mateixa freqüència i crea una llista ordenada amb aquesta informació.
    - Retorna les 10 freqüències més comunes i les 5 menys comunes.
- **Impressió**: Es mostren les 10 freqüències més i menys comunes, indicant quantes paraules tenen cadascuna d'aquestes freqüències.



4. Càlcul de n-grams (ngrams)

- **Funció ngrams**: Genera seqüències de n paraules consecutives (n-grams) del text.
    - Divideix el text en paraules, forma seqüències consecutives de longitud n, i comptabilitza quantes vegades apareix cada n-gram.
    - Retorna una llista ordenada descendentment per freqüència dels n-grams.
- **Exemple d'ús**: Es calculen els 10 trigrammes (seqüències de 3 paraules) més freqüents i es mostren.

```
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
```

---

5. Model d'espai vectorial per calcular la similitud cosinus (cosinesim)

- **Funció cosinesim**: Calcula la similitud cosinus entre dos textos.
    - Les paraules i les seves freqüències es processen sense paraules buides per a cadascun dels textos.
    - Es crea un vector per cada text basat en les freqüències de paraules.
    - La similitud cosinus es calcula com el producte escalar dels vectors dividit pel producte de les seves magnituds (normes).
    - Si una magnitud és 0, la similitud es defineix com 0.
- **Lectura del segon text**: Llegeix el fitxer pg12.txt i calcula la similitud cosinus entre pg11.txt i pg12.txt.
- **Impressió**: Mostra la similitud cosinus entre els dos textos en format decimal amb 4 xifres.

```
val similitud: Double = 0.8764098615744838
pg11.txt - pg12.txt 0,8764

val similitud: Double = 0.9516910222842667
pg11.txt - pg11-net.txt 0,9517

val similitud: Double = 0.962599424114668
pg12.txt - pg12-net.txt 0,9626

val similitud: Double = 0.9889848978642601
pg74.txt - pg74-net.txt 0,9890

val similitud: Double = 0.9713523358626214
pg2500.txt - pg2500-net.txt 0,9714
```

```
val similitud: Double = 0.8443396127082923
pg11 - pg12 amb digrames 0,8443

val similitud: Double = 0.8844339668346107
pg11 - pg12 amb trigrames 0,8844

val similitud: Double = 0.9741393185407533
pg11 - pg11-net amb digrames 0,9741

val similitud: Double = 0.9851456271301621
pg12 - pg12-net amb trigrames 0,9851
```

Part 2

1. **Calcular el nombre promig de referències**

En el primer exercici, per calcular el nombre mitjà de referències, hem definit una funció de mapeig que fa servir ViquipediaParse.parseViquipediaFile i per extreure les referències i associar-les amb la clau "refs". La funció de reducció simplement acumula el nombre total de referències i el divideix pel nombre de fitxers que s'han processat.

```
Hem rebut lencarrec
Going to create MAPPERS!!
All sent to Mappers, now start listening...
All sent to Reducers
All Done from Reducers!
Nombre mitjà de referències: 335,59
```

2. **Donada una query, fer una recomanació fent ús de l'algoritme PR**

En la implementació del PageRank, hem seguit un enfocament iteratiu basat en l'algorisme original. La funció de mapeig calcula el PageRank de cada document, distribuint el seu valor a través dels enllaços entrants segons la fórmula clàssica del PageRank, considerant també la probabilitat de navegació aleatòria. La funció de reducció acumula aquests valors per determinar el nou PageRank dels documents. El procés es repeteix fins que la variació total sigui inferior a un llindar predefinit o s'assoleixi un nombre màxim d'iteracions.

```
// Funció de mapeig per al càlcul del PageRank
def mapPageRank(queryDocs: List[String], links: Map[String, List[String]], d: Double, N: Int): List[(String, Double)] = {
  queryDocs.map { title =>
    val inbound = links.filter(_._2.contains(title)).keys
    val rank = (1 - d) / N + d * inbound.map { prevTitle =>
      val prevRank = 1.0 / N
      prevRank / links(prevTitle).size
    }.sum
    (title, rank) // Retorna el títol amb el seu rank calculat
  }
}

// Funció de reducció per al PageRank
def reducePageRank(key: String, values: List[Double]): (String, Double) = {
  (key, values.sum)
}
```

```
Hem rebut lencarrec
Going to create MAPPERS!!
All sent to Mappers, now start listening...
All sent to Reducers
All Done from Reducers!
Hem rebut lencarrec
Going to create MAPPERS!!
All sent to Mappers, now start listening...
All sent to Reducers
All Done from Reducers!
Títol: Anime, PageRank: 0.0016129032258064518
Títol: Manga, PageRank: 0.0016129032258064518
```

3. **Detectar les pàgines que s'assemblen més**

Per detectar pàgines amb contingut similar, hem implementat una funció de mapeig que compara cada document amb la resta mitjançant la funció cosinesim que teníem de la primera part. Quan la similitud és superior a un llindar establert, s'emmagatzema una parella de títols com a resultat. La funció de reducció acumula aquestes parelles, tenint en compte només les

que no tenen enllaços directes entre elles per evitar redundàncies.

```
val mapFunc = (file: String, docs: List[String]) => {
  val result1 = ViquipediaParse.parseViquipediaFile(file)
  docs.flatMap { doc =>
    val result2 = ViquipediaParse.parseViquipediaFile(doc)
    val sim = cosinesim(result1.contingut, result2.contingut)
    if (sim > threshold && !result1.refs.contains(result2.titol) && !result2.refs.contains(result1.titol))
      // Si la similitud és suficient, tornem la parella de títols
      List((s"${result1.titol} - ${result2.titol}", 1))
    else
      List.empty
  }
}

val reduceFunc = (pair: String, counts: List[Int]) => {
  (pair, counts.sum) // Acumulem les ocurrences de pàgines similars
}
```

```
Or - Mercè Rodoreda i Gurguí
Acer - Shōgun
Mar Mediterrània - Reactor nuclear
Jiddisch - The Great Dictator
25 d'agost - República Democràtica del Congo
Kurt Gödel - Jazz
Esperanto - Independentisme català
Polònia - 24 de maig
Ossètia del Nord - Alània - Vent
Canadà - Sardenya
París - L'Alguer
Història d'Itàlia - Illes Marshall
15 de maig - 25 d'abril
28 de maig - Volga
```

4. **Cal que definiu una funció timeMeasurement[A] | Cal que el MapReduce el parametritzeu amb un cert nombre de mappers**

```
// Funció per mesurar el temps d'execució d'un bloc de codi
def timeMeasurement[A](block: => A): (A, Long) = {
  val startTime = System.nanoTime()
  val result = block
  val endTime = System.nanoTime()
  val elapsedTime = endTime - startTime
  (result, elapsedTime)
}
```

També hem avaluat el rendiment del sistema executant aquestes tasques amb diferents configuracions de mapejadors i reductors: 1, 4, 10 i 20 actors. Utilitzant una funció de mesurament del temps d'execució, hem observat com augmentava l'eficiència del sistema a mesura que incrementàvem el nombre d'actors. En el nostre cas els resultats han estat que hi ha una lleugera diferència a mesura que introduïm actors, però com que per a l'execució hem fet servir un subconjunt dels fitxers .xml la diferència no és massa gran. Si el volum de fitxers és molt gran, introduir més actors pot ser beneficiós i disminuir el temps, però si en posem masses arribarà un moment que serà contraproduent i el fet de desplegar tants actors ens perjudicarà més que el temps que ells mateixos ens estalvien.

```
Temps d'execució amb 1 mapejadors i 1 reductors: 5698,67 ms
Temps d'execució amb 4 mapejadors i 4 reductors: 5379,52 ms
Temps d'execució amb 10 mapejadors i 10 reductors: 5415,96 ms
Temps d'execució amb 20 mapejadors i 20 reductors: 5284,31 ms
```

5. **Tancar correctament el sistema d'actors**

Finalment, per assegurar una correcta gestió dels recursos, hem garantit el tancament del sistema d'actors amb system.terminate().