TD3

1. Etudiez le Fichier. Que pouvez-vous en dire ?

```
//1
// famille arbre.csv:
// Dataset contient 2 colonnes. Une avec les genres d'arbres et une autre avec leurs
      familles
// abrealignementdansparis.csv:
// fichier contenant la position et une description précise de différents arbres dans
paris.
// une union semble être possible entre les deux datasets

**********************************************************************
//Instancier le spark session
val conf = new SparkConf()
  .setMaster("local[4]")
  .setAppName("trees")
  .set("spark.executor.memory", "2g")

val sc = new SparkContext(conf)
```

2. Enlevez le header du fichier, puis Comptez le nombre d'arbres de la ville de Paris

```
val tress = sc
  .textFile("D:/Document D/COURS ESGI/SPARK/arbresalignementparis2010.csv")
  .filter(line ⇒ !line.startsWith("geom"))
  .map(line ⇒ line.split(";"))

 val nb_arbre_paris= tress.count()
 print(nb_arbre_paris)
```

```
2020-01-31 20:11:42,394 - WARN  [main] org.apache.spark.util.SizeEstimator - Failed to check whether UseCompressedOops is set; assuming yes
118447
Process finished with exit code 0
```

3. Enlevez le header d'une autre manière, affichez les 20 premiers types d'arbres (le type d'arbre est représenté par la troisième colonne). Certains arbres n'ont pas de types

```
val first_arbre = tress
   .zipWithIndex()
   .filter(x=> x._2 !=0)
   .map(col => col._1(2))
   .filter(line => line != "")
   .take(20)
first_arbre.foreach(println)
```

```
2020-01-31 20:15:21,397 - WARN  [main] org.apache.spark.util.SizeEstimator
Populus
Platanus
Platanus
Platanus
Platanus
Platanus
Platanus
Platanus
Platanus
Platanus
Platanus
Tilia
Tilia
Tilia
Tilia
Tilia
Tilia
Tilia
Tilia
Tilia

Process finished with exit code 0
```

4. Affichez tous les types d'arbres sans doublons

```
val all_arbre= tress
  .zipWithIndex()
  .filter(x=> x._2 !=0)
  .map(col => col._1(2))
  .filter(line => line != "")
  .distinct()

  all_arbre.foreach(println)
```

```
2020-01-31 20:19:28,789 - WARN  [main] org.apache.spark.util.SizeEstimator
Fagus
Carpinus
Alnus
Clerodendrum
Ligustrum
Taxodium
Koelreuteria
Sterculia
Crataegus
Liriodendron
Parrotia
Hibiscus
Cercis
Betula
Zelkova
Quercus
Prunus
Metasequoia
Magnolia
Ehretia
Pterocarya
Salix
Ptelea
Platanus
Catalpa
Tilia
Aesculus
Paulownia
Morus
Photinia
Populus
Melia
Sorbus
Fraxinus
Cedrela
Sophora
Juglans
Gleditsia
Robinia
Ilex
Diospyros
Celtis
Liquidambar
Corylus
Acer
Amelanchier
Lagerstroemia
Ginkgo
Trachycarpus
Cedrus
Acacia

Process finished with exit code 0
```

5. Trouvez la taille totale de tous les arbres en utilisant .sum(), la taille est représentée par la huitième colonne

```
// 5
    val sum_arbre= tress
      .zipWithIndex()
      .filter(x=> x._2 !=0)
      .map(col => col._1(7).toFloat)
      .filter(line => line != "")
      .sum()
println("question 5:",sum_arbre)
```

```
2020-01-31 20:23:18,566 - WARN  [main] org.apache.spark.util.SizeEstimator -
(question 5:,72138.0)


Process finished with exit code 0
```

6. Trouvez la taille totale de tous les arbres en utilisant .reduce(), la taille est représentée par la huitième colonne

```
//6
val reduce_arbre= tress
  .zipWithIndex()
  .filter(x=> x._2 !=0)
  .map(col => col._1(7).toFloat)
  .reduce((size,a)=>(size+a))
println("question 6:", reduce_arbre)
```

```
2020-01-31 20:27:19,875 - WARN  [main] org.apache.spark.util.SizeEstimator -
(question 6:,72138.0)


Process finished with exit code 0
```

7. Calculez la taille moyenne des arbres. Vous pouvez aussi le faire avec reduce

```
//7

val mean_arbre= tress
  .zipWithIndex()
  .filter(x=> x._2 !=0)
  .map(col => col._1(7).toFloat)
  .mean()

  println("question 7.1:", mean_arbre)

  println("question 7.2 :", reduce_arbre/nb_arbre_paris)
```

```
2020-01-31 20:31:00,949 - WARN  [main] org.apache.spark.util.SizeEstimator -
(question 7.1:,0.6090370295324408)
(question 7.2 :,0.6090319)


Process finished with exit code 0
```

8. Comptez le nombre d'arbres par type en utilisant countByValue

```scala
//8
val type_arbre= tress
  .zipWithIndex()
  .filter(x=> x._2 !=0)
  .map(col => col._1(2))
  .countByValue()
```

```
2020-01-31 20:38:41,007 - WARN  [main] org.apache.spark.util.SizeEstimator - Failed to check whether UseCompressedOops is set; assuming yes
(question 8:,Map(Taxodium -> 1,  -> 21617, Ulmus -> 1212, Ostrya -> 193, Pterocarya -> 656, Clerodendrum -> 18, Gleditsia -> 456, Sophora -> 9137, Quercus -> 890, Tilia -> 9471, Alnus -> 157, I

Process finished with exit code 0
```

```
olea -> 1, Ginkgo -> 535, Zelkova -> 43, Magnolia -> 360, Liriodendron -> 535, Juglans -> 299, Cedrus -> 4, Lagerstroemia -> 151, Betulus -> 2, Malus -> 378, Prunus -> 1377, Trachycarpus -> 51,
```

```
Ehretia -> 3, Melia -> 10, Pyrus -> 1366, Platanus -> 34926, Diospyros -> 36, Celtis -> 2057, Photinia -> 38, Ailanthus -> 466, Parrotia -> 11, Ilex -> 6, Amelanchier -> 103, Ptelea -> 2,
```

```
Acacia -> 3, Ligustrum -> 101, Aesculus -> 14437, Sterculia -> 1, Corylus -> 1492, Cornus -> 19, Koelreuteria -> 167, Albizzia -> 38, Fagus -> 56, Gymnocladus -> 67, Fraxinus -> 2484))
```

9. Comptez le nombre d'arbres par type en utilisant reduceByKey, ordonner le résultat par type d'arbre dans l'ordre alphabétique

```
//9
val red_type_arbre= tress
  .zipWithIndex()
  .filter(x=> x._2 !=0)
  .map(col =>( col._1(2),1))
  .reduceByKey((value, key) => value + key)
  .sortByKey(true)

 red_type_arbre.foreach(println)
```

```
2020-01-31 20:42:53,680 - (Cornus,19)
(,21617)                  (Corylus,1492)
(Lagerstroemia,151)       (Prunus,1377)
(Ligustrum,101)           (Ptelea,2)
(Liquidambar,256)         (Pterocarya,656)
(Acacia,3)                (Crataegus,138)
(Acer,5226)               (Diospyros,36)
(Aesculus,14437)          (Ehretia,3)
(Ailanthus,466)           (Pyrus,1366)
(Albizzia,38)             (Quercus,890)
(Liriodendron,535)        (Robinia,1410)
(Magnolia,360)            (Fagus,56)
(Malus,378)               (Fraxinus,2484)
(Alnus,157)               (Ginkgo,535)
(Amelanchier,103)         (Salix,13)
(Betula,27)               (Sophora,9137)
(Melia,10)                (Sorbus,104)
(Metasequoia,27)          (Gleditsia,456)
(Betulus,2)               (Gymnocladus,67)
(Carpinus,1091)           (Hibiscus,12)
(Catalpa,211)             (Sterculia,1)
(Morus,387)               (Taxodium,1)
(Cedrela,1735)            (Tilia,9471)
(Cedrus,4)                (Ilex,6)
(Ostrya,193)              (Juglans,299)
(Celtis,2057)             (Koelreuteria,167)
(Parrotia,11)             (Trachycarpus,51)
(Paulownia,1016)          (Ulmus,1212)
(Photinia,38)             (Zelkova,43)
(Cercidyphyllum,15)       (olea,1)
```

```
(col._1(0),col._1(1)))    Process finished with exit code 0
```

```
 val fusion = fus_1.join(fus_2)

 val famille_fusion = fusion
   .map(col => col._2._2)
```

10.Jointure avec les familles d'arbres, comptez les arbres par famille,

// Possible de le faire avec moins étape mais une erreur apparaissait.

```
// 10
 val famille = sc
   .textFile("D:/Documen
t D/COURS
ESGI/SPARK/familles
d'arbes.csv")
   .filter(line
⇒ !line.startsWith("geom
"))
   .map(line ⇒
line.split(";"))

val fus_1= tress
  .zipWithIndex()
  .filter(x=> x._2 !=0)
  .map(col
=>( col._1(2),1))

 val fus_2= famille
   .zipWithIndex()
   .filter(x => x._2 !=0)
   .map(col =>
```

```
val family_fus = famille_fusion
    .countByValue()

family_fus.foreach(println)
```

```
2020-01-31 20:50:42,222 - WARN  [main] org.apache.spark.util.SizeEstimator - Failed to check whether UseCompressedOops is set; assuming yes
(Juglandaceae,955)
(Fagales,27)
(Ebenaceae,36)
(Pinaceae,4)
(Rutaceae,2)
(Ginkgoaceae,535)
(Altingiaceae,791)
(Ulmaceae,1255)
(Magnoliaceae,360)
(Scrophulariaceae,1016)
(Cercidiphyllaceae,15)
(Lythraceae,151)
(Hamamelidaceae,11)
(Lamiaceae,18)
(Arecaceae,51)
(Boraginaceae,3)
(Tiliaceae,9471)
(Meliaceae,1745)
(Taxodiaceae,28)
(Bignoniaceae,211)
(Oleaceae,2586)
(Cornaceae,19)
(Moraceae,387)
(Simaroubaceae,466)
(Sapindaceae,19830)
(Aquifoliaceae,6)
(Mimosoideae,38)
(Fabaceae,11158)
(Platanaceae,34926)
(Salicaceae,1273)
(Rosaceae,3504)
(Fagaceae,946)
(Betulaceae,2935)

Process finished with exit code 0
```

11.Jointure avec les familles d'arbres en broadcastant les familles d'arbres, comptez les arbres par famille

```
val familyCollectAsMap= famille
  .map(col => (col(0),col(1)))
  .collectAsMap()
val broadcastFamily = sc.broadcast(familyCollectAsMap)

val custom_tree= tress
  .filter(f=>f(2) != "")
  .map(col =>  broadcastFamily.value(col(2)))
  .countByValue()
 custom_tree.foreach(println)
```

```
2020-01-31 20:56:37,582 - WARN  [main] org.apache.spark.util.SizeEstimator -
(Juglandaceae,955)
(Fagales,27)
(Ebenaceae,36)
(Pinaceae,4)
(Rutaceae,2)
(Ginkgoaceae,535)
(Altingiaceae,791)
(Ulmaceae,1255)
(Magnoliaceae,360)
(Scrophulariaceae,1016)
(Cercidiphyllaceae,15)
(Lythraceae,151)
(Hamamelidaceae,11)
(Lamiaceae,18)
(Arecaceae,51)
(Boraginaceae,3)
(Tiliaceae,9471)
(Meliaceae,1745)
(Taxodiaceae,28)
(Bignoniaceae,211)
(Oleaceae,2586)
(Arecaceae,51)
(Boraginaceae,3)
(Tiliaceae,9471)
(Meliaceae,1745)
(Taxodiaceae,28)
(Bignoniaceae,211)
(Oleaceae,2586)
(Cornaceae,19)
(Moraceae,387)
(Simaroubaceae,466)
(Sapindaceae,19830)
(Aquifoliaceae,6)
(Mimosoideae,38)
(Fabaceae,11158)
(Platanaceae,34926)
(Salicaceae,1273)
(Rosaceae,3504)
(Malvaceae,13)
(Cannabaceae,2057)
(Fagaceae,946)
(Betulaceae,2935)

Process finished with exit code 0
```