Novembre 2017

Exercice 1

Données

```
nom:mcgill,prenom:ben,age:22

nom:smith,prenom:lara,niveau:4

nom:snod,prenom:rick,age:27,niveau:5

nom:kirch,prenom:lars,pays:russia
```

Programme scala

```
val data = sc.textFile(path+"users.txt")

case class attribut(cle:String,valeur:String)

def parseElem(in: String): attribut = {

     val tmp = in.split(":")

     attribut(tmp(0),tmp(1))

     }

parseTuple(in:List[String]) //identique

val parsed = data.map(x=>x.split(",").toList).map(x=>parseTuple(x).sortBy(f=>f.cle))

parsed.collect.foreach(println)

case class attribut(cle:String,pres:Boolean)

def parseElem(in: String): attribut = {

     val tmp = in.split(":")

     attribut(tmp(0),true)

     }

def parseTuple(in:List[String]): List[attribut] = in.map(x=>parseElem(x))

val parsed = data.map(x=>x.split(",").toList).map(x=>parseTuple(x).sortBy(f=>f.cle))
```

```
/*non posée*/

def mergeListAttributes(lat1: List[attribut], lat2: List[attribut]): List[attribut] ={

...}

val synthese = parsed.reduce(mergeListAttributes)

scala> parsed.collect.foreach(println)

List(attribut(age,true), attribut(nom,true), attribut(prenom,true))

List(attribut(niveau,true), attribut(nom,true), attribut(prenom,true))

List(attribut(age,true), attribut(niveau,true), attribut(nom,true), attribut(prenom,true))

List(attribut(nom,true), attribut(pays,true), attribut(prenom,true))
```

Exercice 2 : Algèbre Dataset de Spark

Données

```
n1,p1,m1

m1,p2,n2

n2,p3,n1

n2,p4,m1

n3,p1,n2

n1,isa,t1

n3,isa,t1

m1,isa,t2

m2,isa,tt3
```

Requêtes

```
//preparation

case class Triple(sujet: String, prop: String, objet: String)

val triples = sc.textFile(someFile).

  map(ligne => ligne.split(",")).
```

```
  map(tab => Triple(tab(0), tab(1), tab(2))).toDS()

/*Retourner le nombre de circuits (cycles) de longueur 3, ie les motifs de la forme x → y
→ z → x*/

val t1 =
triples.withColumnRenamed("sujet","x").withColumnRenamed("objet","y").select("x","y")

val t2 =
triples.withColumnRenamed("sujet","y").withColumnRenamed("objet","z").select("y","z")

val t3 =
triples.withColumnRenamed("sujet","z").withColumnRenamed("objet","x1").select("z","x1")

val res = t1.join(t2, "y").join(t3, "z").where("x=x1")

/*On étend l’ensemble de triplets avec des triplets de la forme n, isA, t indiquant que le
noeud n a le type t. Pour simplifier, chaque n a un seul type t.*/

/*Retourner les arcs p communs à au moins deux triplets n, p, n’ et m, p, m’ tels que n et
m ont le même type tandis que n’ et m’ ont deux types distincts.*/

val t_types = triples.where("prop='isa'").select("sujet", "objet")

val t_m = triples.where("prop!='isa'").withColumnRenamed("objet",
"mp").withColumnRenamed("sujet","m")

val t_n = triples.where("prop!='isa'").withColumnRenamed("objet",
"np").withColumnRenamed("sujet","n")

val t_res = t_m.join(t_n, "prop").where("n<m").

join(t_types.withColumnRenamed("sujet", "m"), "m").

join(t_types.withColumnRenamed("sujet","n").withColumnRenamed("objet","o"),"n").where("obj
et=o").select("prop", "np","mp").

join(t_types.withColumnRenamed("sujet","mp"), "mp").

join(t_types.withColumnRenamed("sujet","np").withColumnRenamed("objet",
"o"),"np").where("objet!=o")
```