databricksBDLE_TME2_graphX_21_22_etu

BDLE : GraphX

Description des données synthétiques modélisant un réseau social

- Fichier `facebook_edges_prop.csv` : contient sur chaque ligne les informations suivantes pour un arc:
  `source, destination, type_relation, nombre_messages_échangés`
- Fichier `facebook_users_prop.csv` : contient les informations suivantes pour chaque utilisateur: `utilisateur, prénom, nom, âge`

Lecture des données

Télécharger les fichiers `facebook_edges_prop.csv` et `facebook_users_prop.csv` à partir de cette adresse:

- https://nuage.lip6.fr/s/wZrEiYR4YjQZ9is (https://nuage.lip6.fr/s/wZrEiYR4YjQZ9is)

puis charger ces deux fichiers dans le répertoire `/FileStore/tables/BDLE/TMEGraphes2/` de Databricks.

Initialisation

```
import org.apache.spark.graphx._
import org.apache.spark.rdd.RDD
sc.setLogLevel("ERROR")
```

```
import org.apache.spark.graphx._
import org.apache.spark.rdd.RDD
```

Test Cours

```
type VertexId = Long
val vertices: RDD[(VertexId,(String, String))]= sc.parallelize(Array((3L,
("rxin","student")),(7L,("jgonzal","postdoc")),(5L,("franklin","prof")),(2L,
("istoica", "prof"))))
val edges: RDD[Edge[String]] =
sc.parallelize(Array(Edge(3L,7L,"collab"),Edge(5L, 3L,"advisor" ), Edge(2L,5L,
"colleague"), Edge(5L,7L,"pi")))

val graph = Graph(vertices, edges)
```

```
defined type alias VertexId
vertices: org.apache.spark.rdd.RDD[(VertexId, (String, String))] = ParallelCol
lectionRDD[34] at parallelize at command-4452907426980503:2
edges: org.apache.spark.rdd.RDD[org.apache.spark.graphx.Edge[String]] = Parall
elCollectionRDD[35] at parallelize at command-4452907426980503:3
graph: org.apache.spark.graphx.Graph[(String, String),String] = org.apache.spa
rk.graphx.impl.GraphImpl@4798e4ef
```

```
graph
```

```
res11: org.apache.spark.graphx.Graph[(String, String),String] = org.apache.spa
rk.graphx.impl.GraphImpl@4798e4ef
```

```
graph.vertices.filter { case (id, (name, pos)) => pos == "student" }.count
```

```
res13: Long = 1
```

```
graph.edges.filter(e => e.srcId > e.dstId).count
```

```
res14: Long = 1
```

```
graph.edges.filter { case Edge(src, dst, prop) => src > dst }.count
```

```
res15: Long = 1
```

```
val facts: RDD[String] =
  graph.triplets.map(triplet =>
    triplet.srcAttr._1 + " is the " + triplet.attr + " of " +
triplet.dstAttr._1)
facts.collect.foreach(println(_))
```

```
rxin is the collab of jgonzal
franklin is the advisor of rxin
istoica is the colleague of franklin
franklin is the pi of jgonzal
facts: org.apache.spark.rdd.RDD[String] = MapPartitionsRDD[61] at map at comma
nd-4452907426980510:2
```

```
val inDegrees: VertexRDD[Int] =  graph.inDegrees
```

```
inDegrees: org.apache.spark.graphx.VertexRDD[Int] = VertexRDDImpl[68] at RDD a
t VertexRDD.scala:57
```

```
inDegress.collect.foreach(println(_))
```

```
(3,1)
(5,1)
(7,2)
```

```
def max(a: (VertexId, Int), b: (VertexId, Int)): (VertexId, Int) = {
  if (a._2 > b._2) a else b
}
// Compute the max degrees
val maxInDegree: (VertexId, Int)  = graph.inDegrees.reduce(max)
val maxOutDegree: (VertexId, Int) = graph.outDegrees.reduce(max)
val maxDegrees: (VertexId, Int)   = graph.degrees.reduce(max)
```

```
max: (a: (VertexId, Int), b: (VertexId, Int))(VertexId, Int)
maxInDegree: (VertexId, Int) = (7,2)
maxOutDegree: (VertexId, Int) = (5,2)
maxDegrees: (VertexId, Int) = (5,3)
```

```
val joinRDD: RDD[(VertexId, String)] = sc.parallelize(Array((3L, "étudiant"),
(5L, "professeur")))
val joinedGraph = graph.joinVertices(joinRDD)((id, oldVal, newVal) =>
(oldVal._1, newVal) )
joinedGraph.vertices.collect.foreach(println(_))
```

```
(2,(istoica,prof))
(3,(rxin,étudiant))
(5,(franklin,professeur))
(7,(jgonzal,postdoc))
joinRDD: org.apache.spark.rdd.RDD[(VertexId, String)] = ParallelCollectionRDD
[80] at parallelize at command-4452907426980519:1
joinedGraph: org.apache.spark.graphx.Graph[(String, String),String] = org.apac
he.spark.graphx.impl.GraphImpl@4979f254
```

```
val degreesRDD : VertexRDD[Int] = graph.outDegrees
val joinedGraph: Graph[Int, String] = graph.outerJoinVertices(degreesRDD)((vid,
_, degOpt) => degOpt.getOrElse(0))
```

```
degreesRDD: org.apache.spark.graphx.VertexRDD[Int] = VertexRDDImpl[75] at RDD
at VertexRDD.scala:57
joinedGraph: org.apache.spark.graphx.Graph[Int,String] = org.apache.spark.grap
hx.impl.GraphImpl@52241992
```

```
joinedGraph.vertices.collect.foreach(println(_))
```

```
(2,1)
(3,1)
(5,2)
(7,0)
```

```
val outputGraph: Graph[Double, Double] = joinedGraph.mapTriplets(triplet => 1.0
/ triplet.srcAttr).mapVertices((id, _) => 1.0)
```

```
outputGraph: org.apache.spark.graphx.Graph[Double,Double] = org.apache.spark.g
raphx.impl.GraphImpl@680d4cd4
```

```
outputGraph.triplets.collect.foreach(println(_))
```

```
((3,1.0),(7,1.0),1.0)
((5,1.0),(3,1.0),0.5)
((2,1.0),(5,1.0),1.0)
((5,1.0),(7,1.0),0.5)
```

Exercice 1: Créer le graphe d'utilisateurs

Construire un graphe dirigé (voir la section “Graph Builders” dans la documentation GraphX) à partir des données de cet exercice tel que :

- chaque sommet est identifié par l'attribut utilisateur et ayant pour propriétés les attributs prénom, nom et âge de facebook_users_prop.csv.
- chaque arête reliant deux sommets a pour propriétés les attributs type*relation et nombre_messages*échanés de facebook_edges_prop.csv.

Le type de la structure Spark qui stocke les sommets est le suivant:

- org.apache.spark.rdd.RDD[(org.apache.spark.graphx.VertexId, (String, String, Int))]

Le type de la structure Spark qui stocke les arêtes est le suivant:

- org.apache.spark.rdd.RDD[org.apache.spark.graphx.Edge[(String, Int)]]

Le type de la structure Spark qui stocke le graphe est le suivant:

- org.apache.spark.graphx.Graph[(String, String, Int),(String, Int)]

```
val DATASET_DIR="/FileStore/tables/BDLE/TMEGraphes2/"
var file = "facebook_edges_prop.csv"
var lines = sc.textFile(DATASET_DIR+file, 4)

val edgesList:RDD[Edge[(String, Int)]] = lines.map{s=>
    val parts = s.split(",")
    Edge(parts(0).toLong, parts(1).toLong, (parts(2), parts(3).toInt))
}.distinct()

file = "facebook_users_prop.csv"
lines = sc.textFile(DATASET_DIR+file, 4)

val vertexList:RDD[(VertexId,(String, String, Int))] = lines.map{s=>
    val parts = s.split(",")
    (parts(0).toLong, (parts(1), parts(2), parts(3).toInt))
}.distinct()
val graph = Graph.apply(vertexList, edgesList)
```

```
DATASET_DIR: String = /FileStore/tables/BDLE/TMEGraphes2/
file: String = facebook_users_prop.csv
lines: org.apache.spark.rdd.RDD[String] = /FileStore/tables/BDLE/TMEGraphes2/f
acebook_users_prop.csv MapPartitionsRDD[31] at textFile at command-17282131608
25868:11
edgesList: org.apache.spark.rdd.RDD[org.apache.spark.graphx.Edge[(String, In
t)]] = MapPartitionsRDD[29] at distinct at command-1728213160825868:5
file: String = facebook_users_prop.csv
lines: org.apache.spark.rdd.RDD[String] = /FileStore/tables/BDLE/TMEGraphes2/f
acebook_users_prop.csv MapPartitionsRDD[31] at textFile at command-17282131608
25868:11
vertexList: org.apache.spark.rdd.RDD[(org.apache.spark.graphx.VertexId, (Strin
g, String, Int))] = MapPartitionsRDD[35] at distinct at command-17282131608258
68:13
graph: org.apache.spark.graphx.Graph[(String, String, Int),(String, Int)] = or
g.apache.spark.graphx.impl.GraphImpl@24bdbd1f
```

```
graph.vertices.collect.foreach(println(_))
```

```
(1084,(Nathaniel,Draffen,29))
(3764,(Kenyon,Bohlig,21))
```

```
(3456,(Cory,Hagert,28))
(772,(Danny,Lovinggood,32))
(3272,(Winnie,Hurlbert,45))
(752,(Hebert,Lenberg,43))
(1724,(Vollie,Jenderer,22))
(428,(Sherrill,Stonecypher,33))
(1900,(Lucio,Szabo,45))
(1328,(Maxie,Wessell,20))
(464,(Clem,Moosman,45))
(1336,(Kortney,Blalock,40))
(1040,(Tomas,Niece,26))
(912,(Chauncy,Loseke,36))
(140,(Zettie,Stolpe,20))
(204,(Bolden,Gonzalos,20))
(956,(Geri,Koegel,44))
(1128,(Luther,Petroff,41))
(2892,(Watson,Tomson,28))
(3436,(Gillian,Lundy,30))
(228,(Tarsha,Stanzak,32))
```

Exercice 2

Afficher pour l'utilisateur dont le prénom est 'Kendall' son identifiant, son nom et son âge (utiliser la vue graph.vertices).

```
graph.vertices.filter{case (id, (prenom,nom,age)) => prenom == "Kendall"
}.map{case (id, (prenom,nom,age)) => (id, nom,
age)}.collect.foreach(println(_))
```

```
(2058,Brewbaker,49)
```

```
Réponse attendue: 2058 Brewbaker 49
```

```
   <console>:5: error: identifier expected but integer literal found.
          Réponse attendue: 2058 Brewbaker 49
                            ^
```

Exercice 3

Afficher les prénoms des amis de “Kendall” (utiliser la vue graph.triplets). On considère le graphe non-dirigé.

```
graph.triplets.collect.foreach(println(_))
```

```
((0,(Dalvin,Lairmore,34)),(5,(Mildred,Madras,24)),(acquaintance,94))
((0,(Dalvin,Lairmore,34)),(8,(Nim,Gambaiani,39)),(acquaintance,71))
((0,(Dalvin,Lairmore,34)),(9,(Orelia,Hochstatter,45)),(family,99))
((0,(Dalvin,Lairmore,34)),(11,(Kristian,Evanchalk,25)),(acquaintance,8))
((0,(Dalvin,Lairmore,34)),(16,(Deidra,Lieberg,28)),(family,44))
((0,(Dalvin,Lairmore,34)),(18,(Amberly,Stangle,38)),(friend,3))
((0,(Dalvin,Lairmore,34)),(31,(Brandie,Salvatierra,34)),(friend,53))
((0,(Dalvin,Lairmore,34)),(36,(Ossie,Alouf,37)),(acquaintance,97))
((0,(Dalvin,Lairmore,34)),(39,(Amare,Yakow,24)),(friend,93))
((0,(Dalvin,Lairmore,34)),(41,(Offie,Common,35)),(colleague,68))
((0,(Dalvin,Lairmore,34)),(42,(Jan,Wolski,29)),(family,90))
((0,(Dalvin,Lairmore,34)),(48,(Sheryl,Cernansky,35)),(colleague,66))
((0,(Dalvin,Lairmore,34)),(53,(Darnell,Zange,30)),(friend,31))
((0,(Dalvin,Lairmore,34)),(58,(Kalie,Eskaf,33)),(colleague,29))
((0,(Dalvin,Lairmore,34)),(67,(Daisie,Lamprecht,23)),(acquaintance,33))
((0,(Dalvin,Lairmore,34)),(69,(Dominga,Yanuaria,38)),(friend,73))
((0,(Dalvin,Lairmore,34)),(80,(Donie,Berkovitz,20)),(colleague,69))
((0,(Dalvin,Lairmore,34)),(83,(Mathew,Galindez,35)),(colleague,95))
((0,(Dalvin,Lairmore,34)),(85,(Alfred,Pander,32)),(friend,75))
((0,(Dalvin,Lairmore,34)),(89,(Ammie,Scalzi,37)),(friend,80))
((0,(Dalvin,Lairmore,34)),(92,(Jean,Hunzeker,36)),(friend,40))
```

```
graph.triplets.filter{ triplet => ( triplet.attr._1 == "friend" &&
(triplet.srcAttr._1  == "Kendall" || triplet.dstAttr._1  == "Kendall" )
)}.map{
                triplet  => if (triplet.srcAttr._1==  "Kendall")
(triplet.dstAttr._1) else  (triplet.srcAttr._1)
}.collect.foreach(u=>println(u))
```

```
Toccara
Carroll
Kamren
Con
Jonas
Tamia
Delsie
Verle
Ancel
Zakary
Anais
Barnard
Norma
Christian
```

```
Elzie
Anya
Wanita
Alfredo

Réponse attendue:
Veda
Hilbert
Darl
Toccara
Carroll
Tammi
Shenna
...
True
Wanita
Alfredo
Donnie
```

Exercice 4

Afficher les identifiants des utilisateurs qui ont désigné “Kendall” comme collègue et qui ont échangé plus de 70 messages avec lui (utiliser la vue graph.triplets).

```
"""graph.triplets.filter{triplet => ( triplet.attr._1 == "colleague" &&
triplet.attr._2 >= 70   && (triplet.srcAttr._1  == "Kendall" ||
triplet.dstAttr._1  == "Kendall" )  )}.map(triplet => if (triplet.srcAttr._1==
"Kendall") (triplet.dstId) else  (triplet.srcId)
).distinct().collect().take(10).foreach(x=>println(x))
"""
```

```
graph.triplets.filter{ triplet => ( triplet.attr._1 == "colleague" &&
triplet.attr._2 >= 70   && triplet.dstAttr._1  == "Kendall" ) }.map{triplet =>
(triplet.srcId) }.distinct().collect().take(10).foreach(x=>println(x))
```

```
1941
1966
1983
command-1728213160825876:1: warning: a pure expression does nothing in stateme
nt position; multiline expressions may require enclosing parentheses
"""graph.triplets.filter{triplet => ( triplet.attr._1 == "colleague" && triple
```

```
t.attr._2 >= 70   && (triplet.srcAttr._1  == "Kendall" || triplet.dstAttr._1
== "Kendall" )  )}.map(triplet => if (triplet.srcAttr._1==  "Kendall") (triple
t.dstId) else  (triplet.srcId) ).distinct().collect().take(10).foreach(x=>prin
tln(x))
^


Réponse attendue: 1966, 1983, 1941
```

```
   <console>:5: error: identifier expected but integer literal found.
          Réponse attendue: 1966, 1983, 1941
                            ^
```

Exercice 5

Afficher l'identifiant de l'utilisateur qui a désigné le plus d'utilisateurs comme amis et avec lesquels il a échangé plus de 80 messages. Afficher également le nombre de ces amis (vous pouvez utiliser la méthode Array.maxBy).

```
graph.edges.filter{case Edge(src,dst,prop) => (prop._2 > 80 && prop._1 ==
"friend")}.map{case Edge(src,dst,prop) =>
(src,1)}.reduceByKey(_+_).collect.maxBy(_._2)


res7: (org.apache.spark.graphx.VertexId, Int) = (107,59)


Réponse attendue:
res4: (org.apache.spark.graphx.VertexId, Int) = (107,59)
```

```
   <console>:6: error: ';' expected but ':' found.
          res4: (org.apache.spark.graphx.VertexId, Int) = (107,59)
              ^
```

Exercice 6

Afficher le nom, le prénom et l'âge des voisins de “Kendall”, en considérant que le graphe est non-dirigé (utiliser la vue graph.triplets) suivant les trois manières suivantes :

- En utilisant graph.triplets seulement
- En utilisant collectNeighbors
- En utilisants aggregateMessages

```
graph.triplets.collect.foreach(println(_))
```

```
((0,(Dalvin,Lairmore,34)),(5,(Mildred,Madras,24)),(acquaintance,94))
((0,(Dalvin,Lairmore,34)),(8,(Nim,Gambaiani,39)),(acquaintance,71))
((0,(Dalvin,Lairmore,34)),(9,(Orelia,Hochstatter,45)),(family,99))
((0,(Dalvin,Lairmore,34)),(11,(Kristian,Evanchalk,25)),(acquaintance,8))
((0,(Dalvin,Lairmore,34)),(16,(Deidra,Lieberg,28)),(family,44))
((0,(Dalvin,Lairmore,34)),(18,(Amberly,Stangle,38)),(friend,3))
((0,(Dalvin,Lairmore,34)),(31,(Brandie,Salvatierra,34)),(friend,53))
((0,(Dalvin,Lairmore,34)),(36,(Ossie,Alouf,37)),(acquaintance,97))
((0,(Dalvin,Lairmore,34)),(39,(Amare,Yakow,24)),(friend,93))
((0,(Dalvin,Lairmore,34)),(41,(Offie,Common,35)),(colleague,68))
((0,(Dalvin,Lairmore,34)),(42,(Jan,Wolski,29)),(family,90))
((0,(Dalvin,Lairmore,34)),(48,(Sheryl,Cernansky,35)),(colleague,66))
((0,(Dalvin,Lairmore,34)),(53,(Darnell,Zange,30)),(friend,31))
((0,(Dalvin,Lairmore,34)),(58,(Kalie,Eskaf,33)),(colleague,29))
((0,(Dalvin,Lairmore,34)),(67,(Daisie,Lamprecht,23)),(acquaintance,33))
((0,(Dalvin,Lairmore,34)),(69,(Dominga,Yanuaria,38)),(friend,73))
((0,(Dalvin,Lairmore,34)),(80,(Donie,Berkovitz,20)),(colleague,69))
((0,(Dalvin,Lairmore,34)),(83,(Mathew,Galindez,35)),(colleague,95))
((0,(Dalvin,Lairmore,34)),(85,(Alfred,Pander,32)),(friend,75))
((0,(Dalvin,Lairmore,34)),(89,(Ammie,Scalzi,37)),(friend,80))
((0,(Dalvin,Lairmore,34)),(92,(Jean,Hunzeker,36)),(friend,40))
```

```
graph.aggregateMessages[Array[String]]( { triplet =>
  if (triplet.dstAttr._1 == "Kendall")
  triplet.sendToDst(Array(triplet.srcAttr._1))
  if (triplet.srcAttr._1 == "Kendall")
  triplet.sendToSrc(Array(triplet.dstAttr._1))},

   (a,b)  => (a ++ b) ).values.collect.foreach(t => println(t.size))
```

```
76
```

```
Résultat attendu: 76 résultats
Veda Kustra 40
Hilbert Coakley 25
Darl Borr 42
Toccara Fahlsing 45
Carroll Foronda 37
Tammi Kuske 30
Shenna Blasing 26
Vivian Ugolini 45
Norwood Kirwan 29
Izabella Macrostie 47
Effie Andujo 41
Kamren Gesualdo 43
...
True Ficke 42
Wanita Purslow 32
Alfredo Izard 20
Donnie Lengyel 41
```

```
//1. En utilisant graph.triplets seulement
graph.triplets.filter{ triplet => (triplet.srcAttr._1=="Kenddall" ||
triplet.dstAttr._1=="Kendall")}.map{triplet =>
(triplet.srcAttr,triplet.dstAttr)}.collect.foreach{case ((p,n,a), (p1,n1,a1))
=> if(p=="Kendall") println(p1+" "+n1+" "+a1) else println(p+" "+n+" "+a)}
```

```
Veda Kustra 40
Hilbert Coakley 25
Solomon McCullum 32
Con Ozga 25
Jonas Muzzillo 41
Jeryl Fridman 27
Major Verhey 35
Nicole Zorich 41
Shellie Sivyer 40
Brisa Parkers 33
Scott Fiallos 23
Verle Huhman 27
Chaz Rathe 47
Karyn Grix 26
Suzan Goslin 29
```

```
//2. En utilisant collectNeighbors
graph.collectNeighbors(EdgeDirection.Either).innerJoin(graph.vertices.filter{ca
se (id,(p,n,a)) => p=="Kendall"}) {case (id, list, attrs) =>
list}.values.collect.foreach(list => (list.foreach(sommet =>
print(sommet._2)),println(_)))
```

```
(Veda,Kustra,40)(Hilbert,Coakley,25)(Darl,Borr,42)(Toccara,Fahlsing,45)(Carrol
l,Foronda,37)(Tammi,Kuske,30)(Shenna,Blasing,26)(Vivian,Ugolini,45)(Norwood,Ki
rwan,29)(Izabella,Macrostie,47)(Effie,Andujo,41)(Kamren,Gesualdo,43)(Kristian,
Durrant,35)(Abagail,Discon,37)(Kerrie,Rattler,40)(Idabelle,Hightree,20)(Solomo
n,McCullum,32)(Con,Ozga,25)(Jonas,Muzzillo,41)(Tamia,Guecho,30)(Delsie,Maphis,
34)(Emile,Stusse,42)(Reece,Postle,43)(Hanson,Torralba,28)(Richie,Clow,44)(Shau
na,Keiswetter,39)(Schuyler,Magrath,38)(Wally,Estimable,35)(Wyatt,Cate,22)(Jery
l,Fridman,27)(Major,Verhey,35)(Nicole,Zorich,41)(Shellie,Sivyer,40)(Brisa,Park
ers,33)(Scott,Fiallos,23)(Verle,Huhman,27)(Ancel,Schaap,34)(Harriett,Wieseman,
30)(Ova,Garavaglia,44)(Verla,Taube,40)(Yuridia,Lofgreen,46)(Lilia,Bueter,31)(A
donis,Kosik,23)(Erwin,Burch,43)(Darrick,Engert,44)(Zakary,Slaugh,27)(Anais,Gro
well,35)(Miguelangel,Dang,43)(Barnard,Bruski,26)(Keyshawn,Pippenger,48)(Franci
sco,Hagaman,46)(Syble,Martine,47)(Walton,Woodfield,40)(Mackenzie,Calvey,48)(Pr
eston,Karoly,44)(Adela,Bouquet,21)(Deana,Dierolf,28)(Susanne,Grinie,36)(Chaz,R
athe,47)(Karyn,Grix,26)(Suzan,Goslin,29)(Alpheus,Arambuia,40)(Norma,Romanek,4
2)(Landon,Bussert,21)(Christian,Marcinkiewicz,47)(Loree,Demaris,48)(Elzie,Melo
tt,35)(Fernanda,Cashatt,28)(Sheyla,Nozick,34)(Anya,Leadbeater,30)(Christina,St
reed,30)(Glenn,Figart,26)(True,Ficke,42)(Wanita,Purslow,32)(Alfredo,Izard,20)
(Donnie,Lengyel,41)
```

```
//3. En utilisants aggregateMessages

graph.aggregateMessages[Array[String]] (
triplet => {
if(...)
triplet.sendToDst(...)
if(...)
triplet.sendToSrc(...)
},
(a,b) =>
).values.collect.foreach(...)
```

```
   <console>:9: error: illegal start of simple expression
          if(...)
             ^
   <console>:13: error: ')' expected but '}' found.
          },
          ^
```

```
<console>:15: error: illegal start of simple expression
       ).values.collect.foreach(...)
       ^
```

Exercice 7

Afficher le nombre de liens entrants minimum du graphe. Afficher les noms et les id des utilisateurs qui ont le nombre minimum de liens entrants, les utilisateurs sans liens entrants ne seront pas affichés (utiliser graph.inDegrees).

```
var minDeg = graph.inDegrees.values.min
graph.inDegrees.filter{case (id,d) => d==minDeg}.innerJoin(graph.vertices){case
(id, d, u) => (u._1)}.
collect.foreach(println(_))
```

```
(956,Geri)
(160,Adelle)
(2664,Chantal)
(2712,Leandra)
(1176,Courtney)
(2732,Felix)
(292,Bell)
(4008,Christiana)
(4,Kimberely)
(3452,Yaritza)
(3980,Devan)
(2668,Junie)
(1200,Moody)
(120,Charlie)
(2700,Refugio)
(360,Kia)
(52,Florian)
(64,Olena)
(3492,Elgie)
(3856,Darwyn)
(692,Verl)
```

```
Résultat attendu:
956 Name  Geri
160 Name  Adelle
2664 Name  Chantal
2712 Name  Leandra
1176 Name  Courtney
2732 Name  Felix
......
3459 Name  Hetty
951 Name  Donnell
687 Name  Bunk
47 Name  Arline
3503 Name  Annamae
minDeg: Int = 1
```

Exercice 8

Afficher les noms des utilisateurs qui n'ont pas de liens entrants (utiliser inDegrees et outerJoinVertices).

```
graph.outerJoinVertices(graph.inDegrees){
case (id,p,ind) => (p._1,ind.getOrElse(0))
}.vertices.filter{case (id,(p,ind)) =>
ind==0}.collect.foreach(x=>println(x._2._1 ,"indegree",x._2._2))
```

```
(Dalvin,indegree,0)
(Floy,indegree,0)
```

```
Résultat attendu:
Dalvin indegree  0
Floy indegree  0
```

Exercice 9

Afficher les noms des utilisateurs dont le nombre de liens entrants, ainsi que le nombre de de liens sortants est 5. Affichez également le nombre de liens entrants pour chaque utilisateur obtenu.

```
graph.outerJoinVertices(graph.inDegrees){
case (id, prop, opt) => (prop._1,opt.getOrElse(0))
}.outerJoinVertices(graph.outDegrees){
case (id, prop, opt) => (prop._1, prop._2 , opt.getOrElse(0))
}.vertices.filter{case (id,(nom,ind,outd)) => ind==5 &&
outd==5}.collect.foreach(x=>println(x._2._1, "indegrees",x._2._2))
```

```
(Pauline,indegrees,5)
(Rae,indegrees,5)
(Lewis,indegrees,5)
(Cyril,indegrees,5)
(Billie,indegrees,5)
(Arely,indegrees,5)
(Krish,indegrees,5)
(Hessie,indegrees,5)
(Masao,indegrees,5)
(Archibald,indegrees,5)
(Rosemary,indegrees,5)
(Harlow,indegrees,5)
(General,indegrees,5)
(Letitia,indegrees,5)
```

```
Résultat attendu:
Pauline 5
Rae 5
Lewis 5
Cyril 5
Billie 5
Arely 5
Krish 5
Hessie 5
Masao 5
Archibald 5
Rosemary 5
Harlow 5
General 5
Letitia 5
```

Exercice 10

Pour les utilisateurs Dalvin, Kendall et Elena affichez les prénoms et l'âge des utilisateurs les plus âgés parmi les utilisateurs qui les ont désigné comme amis. Si un utilisateur parmi les trois n'a pas de liens entrants le programme doit afficher un message.

```
val names = List("Dalvin", "Kendall" ,"Elena")
val oldest = graph.aggregateMessages[(String, Int)](
triplet => { // Map Function

// Send message to destination vertex containing counter and age

 if (names.contains(triplet.dstAttr._1) && (triplet.attr._1 == "friend"))
 triplet.sendToDst((triplet.srcAttr._1,triplet.srcAttr._3))
},
(a,b) => if (a._2 > b._2) a else b


)

graph.vertices.filter{case (id,(n,p,a)) => names.contains(n)}.leftJoin(oldest)
{ (id, user, optOldestFollower) =>
optOldestFollower match {
case None => user._1 + " has not fiends followsers "
case Some((name, age)) => name + " aged " + age + " is the oldest follower of
"+ user._1
}
}.collect.foreach {x => println( x._2)}
```

```
Dalvin has not fiends followsers
Jonas aged 41 is the oldest follower of Kendall
River aged 22 is the oldest follower of Elena
names: List[String] = List(Dalvin, Kendall, Elena)
oldest: org.apache.spark.graphx.VertexRDD[(String, Int)] = VertexRDDImpl[442]
at RDD at VertexRDD.scala:57
```

```
Résultat attendu:
Dalvin does not have any followers.
Jonas aged 41 is the oldest follower of Kendall.
River aged 22 is the oldest follower of Elena.
```

Exercice 11

Afficher les identifiants des utilisateurs qui ont désigné Kendall et Lilia comme étant leurs amis (amis communs de Kendall et de Lilia).

```
val names = List("Lilia", "Kendall" )
val com = graph.aggregateMessages[Int] (
triplet =>{

 if (triplet.dstAttr._1 == "Lilia" && (triplet.attr._1 == "friend"))
 triplet.sendToSrc(1)
 if (triplet.dstAttr._1 == "Kendall" && (triplet.attr._1 == "friend"))
  triplet.sendToSrc(1)
},
(a,b) => a + b
  ).filter{case (id,nba)=> nba ==2}.collect.foreach(u=>println(u._1))
```

```
2020
1943
names: List[String] = List(Lilia, Kendall)
com: Unit = ()
```

```
Résultat attendu:
2020
1943
```

Exercice 12

Pour chaque utilisateur, trouver l'âge moyen de ses amis (on considère que le graphe est non-dirigé). Utiliser la méthode aggregateMessages.

Rappel Le graphe étant dirigé, il faudra s'assurer que tous les sommets envoient des messages à leurs successeurs ainsi qu'à leurs prédécesseurs.

```
val ages = graph.aggregateMessages[(Int, Double)]( //on obtient un nouveau
graphe (ages)
triplet => { // Map Function
triplet.sendToSrc(1,triplet.dstAttr._3)
triplet.sendToDst(1,triplet.srcAttr._3)
},
(a, b) => (a._1+b._1,a._2+b._2)
)
ages.mapValues(x=> x._2/x._1 ).collect.foreach(println(_))
```

```
(1084,29.466666666666665)
(3764,36.55172413793103)
(3456,34.627450980392155)
(772,33.77777777777778)
(3272,34.56818181818182)
(752,36.13157894736842)
(1724,36.23728813559322)
(428,35.208695652173915)
(1900,36.65384615384615)
(1328,32.9)
(464,35.8)
(1336,35.472727272727276)
(1040,36.19402985074627)
(912,32.333333333333336)
(140,32.90909090909091)
(204,35.59090909090909)
(956,32.5)
(1128,35.326530612244895)
(2892,36.03448275862069)
(3436,38.142857142857146)
(228,32.0)
```

Résultat attendu:

```
(1084,29.466666666666665)
(3764,36.55172413793103)
(3456,34.627450980392155)
(772,33.77777777777778)
(3272,34.56818181818182)
(752,36.13157894736842)
(1724,36.23728813559322)
(428,35.208695652173915)
(1900,36.65384615384615)
(1328,32.9)
(464,35.8)
....
(3203,36.0)
(2967,35.13636363636363)
(155,37.666666666666664)
(147,34.5)
```

Exercice 13

Calculer pour chaque noeud la liste des différentes types de relations sur ses arcs sortants. L'attribut d'un noeud contenant le type de ses liens sortants sera de type Set[String]. Construire ensuite un nouveau graphe où chaque noeud a comme attributs son prénom et la liste des types de ses arcs sortants, ou une liste vide s'il n'a pas d'arcs sortants (utiliser outerJoinVertices). Afficher la liste des types des arcs sortants pour l'utilisateur Elena dans ce nouveau graphe.

```
val tarcs = graph.aggregateMessages[Set[String]](
triplet => {
triplet.sendToSrc(Set(triplet.attr)
},
(a,b) => a ++ b
)
graph.outerJoinVertices(tarcs){case (id,prop,tracs )= >
 match{
case Some(typeOpt) => (id,prop._1,tracs)
case None => (id,prop._1,null)
}
}.collect.foreach(println(_))
```

```
   <console>:8: error: ')' expected but '}' found.
          },
          ^
   <console>:11: error: '=>' expected but '=' found.
          graph.outerJoinVertices(tarcs){case (id,prop,tracs )= >
                                                              ^
```

```
Résultat attendu:
(3647,(Elena,Set(acquaintance, friend)))
```

Exercice 14

Affichez les prénoms des amis des amis de Deana (à distance 2 de Deana), pour un graphe dirigé sans utiliser la fonction pregel (suggestion: utilisez graph.aggregateMessages et graph.outerJoinVertices).

```
val neighs = graph.aggregateMessages[Boolean] (
triplet => {
if(...)
...
},
(a,b) =>...
)
graph.outerJoinVertices(neighs){...}.triplets.filter{...}.
map(...).collect.foreach(println(_))
```

```
Résultat attendu:
Sharita
Darrick
Idabelle
```

Exercice 15

Affichez les prénoms des amis des amis de Deana (à distance 2 de Deana), pour un graphe dirigé en utilisant la fonction pregel. Vous pouvez prendre comme point de départ l'implantation de l'algorithme des plus courts chemins dans la section “Pregel API”.