# Assignment 6 - GraphX

1. Launch the Spark shell.

```
spark-shell
```

2. Import the GraphX libs, which are `org.apache.spark.graphx._` and `org.apache.spark.rdd.RDD`.

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

3. Build the property graph shown in Figure 1.

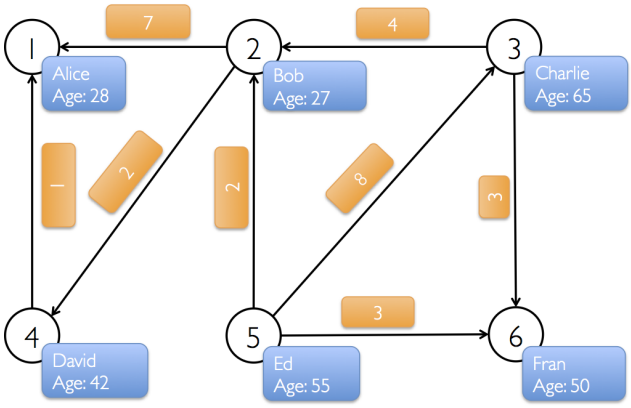


Figure 1: The property graph.

```scala
val vertexArray = Array(
  (1L, ("Alice", 28)),
  (2L, ("Bob", 27)),
  (3L, ("Charlie", 65)),
  (4L, ("David", 42)),
  (5L, ("Ed", 55)),
  (6L, ("Fran", 50))
  )

val edgeArray = Array(
  Edge(2L, 1L, 7),
  Edge(2L, 4L, 2),
  Edge(3L, 2L, 4),
  Edge(3L, 6L, 3),
  Edge(4L, 1L, 1),
  Edge(5L, 2L, 2),
  Edge(5L, 3L, 8),
  Edge(5L, 6L, 3)
  )

val vertexRDD: RDD[(Long, (String, Int))] = sc.parallelize(vertexArray)
val edgeRDD: RDD[Edge[Int]] = sc.parallelize(edgeArray)
val graph: Graph[(String, Int), Int] = Graph(vertexRDD, edgeRDD)
```

4. Display the name of the users older than 30 years old.

```scala
// Solution 1
graph.vertices.filter { case (id, (name, age)) => age > 30 }.foreach {
  case (id, (name, age)) => println(s"$name is $age")
}

// Solution 2
graph.vertices.filter(_._2._2 > 30).foreach(v => println(s"${v._2._1} is ${v._2._2}"))

// Solution 3
for ((id, (name, age)) <- graph.vertices.filter(_._2._2 > 30).collect)
  println(s"$name is $age")
```

5. Display the in-degree of each vertex.

```scala
val inDegrees: VertexRDD[Int] = graph.inDegrees
inDegrees.foreach(println)
```

6. Display who follows who (through the edges direction).

```scala
/**
  * Triplet has the following Fields:
  *   triplet.srcAttr: (String, Int)
  *   triplet.dstAttr: (String, Int)
  *   triplet.attr: Int
  *   triplet.srcId: VertexId
  *   triplet.dstId: VertexId
  */

graph.triplets.foreach(t => println(s"${t.srcAttr._1} follows ${t.dstAttr._1}"))
```

7. Display who likes who (if the edge value is greater than 5).

```
graph.triplets.filter(_.attr > 5).foreach(t =>
  println(s"${t.srcAttr._1} likes ${t.dstAttr._1}"))
```

8. Make a user graph such that each vertex stores the number of its incoming and outgoing links.

```
case class User(name: String, age: Int, inDeg: Int, outDeg: Int)

// Create a user Graph
val initialUserGraph: Graph[User, Int] = graph.mapVertices {
  case (id, (name, age)) => User(name, age, 0, 0)
}

// Fill in the degree information
/**
  * def outerJoinVertices[U, VD2](other: RDD[(VertexID, U)])
  *    (mapFunc: (VertexID, VD, Option[U]) => VD2)
  *  : Graph[VD2, ED]
  */
val userGraph = initialUserGraph.outerJoinVertices(initialUserGraph.inDegrees) {
  case (id, u, inDegOpt) => User(u.name, u.age, inDegOpt.getOrElse(0), u.outDeg)
}.outerJoinVertices(initialUserGraph.outDegrees) {
  case (id, u, outDegOpt) => User(u.name, u.age, u.inDeg, outDegOpt.getOrElse(0))
}

// Display the userGraph
userGraph.vertices.foreach { case (id, u) => println(s"User $id is called
  ${u.name} and is followed by ${u.inDeg} people.")
}
```

9. Display the name of the users who are followed by the same number of people they follow. For example Bob follows two persons, and two persons follow Bob.

```
userGraph.vertices.filter { case (id, u) => u.inDeg == u.outDeg }
  .foreach { case (id, u) => println(u.name) }
```

10. Display the oldest follower for each user.

```
/**
  * def mapReduceTriplets[MsgType](
  * // Function from an edge triplet to a collection of messages (i.e., Map)
  * map: EdgeTriplet[VD, ED] => Iterator[(VertexId, MsgType)],
  * // Function that combines messages to the same vertex (i.e., Reduce)
  * reduce: (MsgType, MsgType) => MsgType)
  * : VertexRDD[MsgType]
  */
val oldestFollower: VertexRDD[(String, Int)] = userGraph.mapReduceTriplets[(String, Int)]
  (edge => Iterator((edge.dstId, (edge.srcAttr.name, edge.srcAttr.age))),
  (a, b) => if (a._2 > b._2) a else b)

userGraph.vertices.leftJoin(oldestFollower) { (id, user, optOldestFollower) =>
  optOldestFollower match {
    case None => s"${user.name} does not have any followers."
    case Some((name, age)) => s"${name} is the oldest follower of ${user.name}."
  }
}.foreach { case (id, str) => println(str)
```

11. Find the average age of the followers of each user.

```
val averageAge: VertexRDD[Double] = userGraph.mapReduceTriplets[(Int, Double)] (
  edge => Iterator((edge.dstId, (1, edge.srcAttr.age.toDouble))),
  (a, b) => ((a._1 + b._1), (a._2 + b._2))
  ).mapValues((id, p) => p._2 / p._1)

scala> userGraph.vertices.leftJoin(averageAge) { (id, user, optAverageAge) =>
  optAverageAge match {
    case None => s"${user.name} does not have any followers."
    case Some(avgAge) => s"The average age of ${user.name}\'s followers is $avgAge."
  }
}.foreach { case (id, str) => println(str) }
```

12. Make a subgraph of the users that are 30 or older.

```
val olderGraph = userGraph.subgraph(vpred = (id, u) => u.age >= 30)
```

13. Compute the connected components and display the component id of each user.

```
val cc = olderGraph.connectedComponents
olderGraph.vertices.leftJoin(cc.vertices) {
  case (id, u, comp) => s"${u.name} is in component ${comp.get}"
}.foreach{ case (id, str) => println(str) }
```

14. Write a standalone application to make a graph from the *followers file* (shown below), measure the page rank of the graph, and display the page rank of each vertex along with its user name. Each row of the *users* file is assigning a name to a vertex.

*followers file*:
2 1
4 1
1 2
6 3
7 3
7 6
6 7
3 7

*users:*
1,Seif,SICS
2,Amir,SICS
3,Jim,KTH
4,Ahmad,SICS
6,Vlad,KTH
7,Fatemeh,SICS
8,Anonsys

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

object Pagerank {
  def main(args: Array[String]) {
    val sc = new SparkContext("local", "PageRank",
      "127.0.0.1", List("target/scala-2.10/pagerank_2.10-1.0.jar"))
    val graph = GraphLoader.edgeListFile(sc, "../data/followers")

    // Run PageRank
    val ranks = graph.pageRank(0.0001).vertices

    // Join the ranks with the usernames
    val users = sc.textFile("../data/users").map { line =>
      val fields = line.split(",")
      (fields(0).toLong, fields(1))
    }

    val ranksByUsername = users.join(ranks).map {
      case (id, (username, rank)) => (username, rank)
    }

    // Print the result
    println(ranksByUsername.collect().mkString("\n"))
  }
}
```

```
// simple.sbt:
name := "PageRank"

version := "1.0"

scalaVersion := "2.10.3"

libraryDependencies ++= Seq(
    "org.apache.spark" %% "spark-core" % "0.9.0-incubating",
    "org.apache.spark" %% "spark-graphx" % "0.9.0-incubating"
)

resolvers += "Akka Repository" at "http://repo.akka.io/releases/"
```