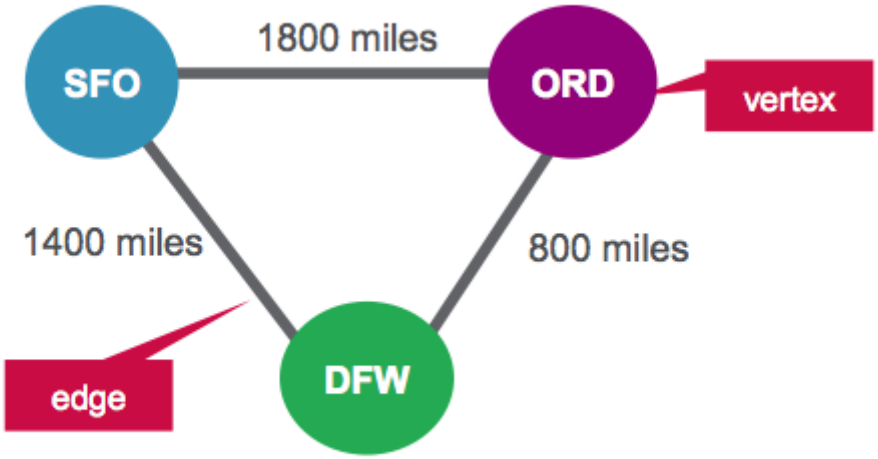
EJERCICIO1 DE AEROPUERTOS

Un **gráfico dirigido** es un gráfico donde los bordes tienen una dirección asociada a ellos. Un ejemplo de un gráfico dirigido es un seguidor de Twitter. El usuario Bob puede seguir al usuario Carol sin implicar que el usuario Carol siga al usuario Bob.




```
import org.apache.spark._
import org.apache.spark.rdd.RDD
// import classes required for using GraphX
import org.apache.spark.graphx._
object Ejemplo {

 def main(args: Array[String]): Unit = {
  val conf = new SparkConf().setAppName("primera").setMaster("local[*]")
  val sc = new SparkContext(conf)

  // create vertices RDD with ID and Name

  val vertices=Array((1L, ("MADRID")),(2L, ("LONDRES")),(3L,("SEVILLA")))
  val vRDD= sc.parallelize(vertices)
  vRDD.take(1)

  // Defining a default vertex called nowhere
  val nowhere = "nowhere"

  // create routes RDD with srcid, destid, distance
  val edges = Array(Edge(1L,2L,1800),Edge(2L,3L,800),Edge(3L,1L,1400))
  val eRDD= sc.parallelize(edges)

  eRDD.take(2)
```

```
   // define the graph
   val graph = Graph(vRDD,eRDD, nowhere)

// graph vertices
   graph.vertices.collect.foreach(println)


   // graph edges
   graph.edges.collect.foreach(println)


//graph.numVertices
   print("numero de vertices"+graph.vertices.count)
   //Number of Edges

   print("el numero de  aristas es"+graph.edges.count)
   // How many airports?
   print("numero de aeropuerto")
   val numairports = graph.numVertices
   println(numairports)

}}
```

EJERCICIO2 AEROPUERTO CON OPERACIONES Y FILTRADO

```
import org.apache.spark._

import org.apache.spark.rdd.RDD

// import classes required for using GraphX

import org.apache.spark.graphx._

object Ejemplo {


  def main(args: Array[String]): Unit = {

   val conf = new SparkConf().setAppName("primera").setMaster("local[*]")

   val sc = new SparkContext(conf)


   // create vertices RDD with ID and Name


   val vertices=Array((1L, ("MADRID")),(2L, ("LONDRES")),(3L,("SEVILLA")))
```

```
val vRDD= sc.parallelize(vertices)
vRDD.take(1)
// Array((1,SFO))

// Defining a default vertex called nowhere
val nowhere = "nowhere"

// create routes RDD with srcid, destid, distance
val edges = Array(Edge(1L,2L,1800),Edge(2L,3L,800),Edge(3L,1L,1400))
val eRDD= sc.parallelize(edges)

eRDD.take(2)

// define the graph
val graph = Graph(vRDD,eRDD, nowhere)
// graph vertices
graph.vertices.collect.foreach(println)

// graph edges
graph.edges.collect.foreach(println)
// NUMERO DE  ARISTAS QUE SALEN DE UN VERTICE
val inDegrees: VertexRDD[Int] = graph.inDegrees
 inDegrees.collectAsMap().foreach(println)

//OPERACIONES****************************************
  // routes > 1000 miles distance?
graph.edges.filter { case Edge(origen, destino, prop) => prop > 1000
}.collect.foreach(println)
```

/* La clase EdgeTriplet extiende la clase Edge al agregar los miembros srcAttr y dstAttr que contienen las propiedades de origen y destino, respectivamente.*/

graph.triplets.take(3).foreach(println)

/*((1,MADRID),(2,LONDRES),1800)

((2,LONDRES),(3,SEVILLA),800)

((3,SEVILLA),(1,MADRID),1400)*/

// distancia entre los aeropuertos y ordenado de mayor a menor distancia

graph.triplets.sortBy(_.attr, ascending=false).map(triplet =>

"Distance " + triplet.attr.toString + " from " + triplet.srcAttr + " to " + triplet.dstAttr + ".").collect.foreach(println)

//

}

}

EJERCICIO3 TENEMOS UNA SERIE DE USUARIOS QUE que mantienen relaciones entre ellos

EL VERTICE SERA EL NOMBRE Y LA ARISTA LA RELACIÓN QUE MANTIENEN ENTRE ELLOS

```
import org.apache.spark.graphx.{Edge, Graph}
import org.apache.spark.rdd.RDD
import org.apache.spark.{SparkConf, SparkContext}
import org.apache.spark.SparkContext
import org.apache.spark.sql.SparkSession
import org.apache.spark.graphx._
import org.apache.spark.graphx.GraphLoader


object Ejemplo {
  def main(args: Array[String]): Unit = {

      val conf = new SparkConf().setAppName("primera").setMaster("local[*]")
      val sc = new SparkContext(conf)
```

```
      val personas: RDD[(VertexId, (String, String))] = sc.parallelize(Array((3L, ("pepe",
"estudiante")), (7L, ("jgonzal", "postdoctor")),
     (5L, ("fran", "profesor")), (2L, ("Lisa", "profesor"))))
   // Create an RDD for edges
   val relaciones: RDD[Edge[String]] = sc.parallelize(Array(Edge(3L, 7L, "colabora"),
Edge(5L, 3L, "aconseja"),
     Edge(2L, 5L, "ayuda"), Edge(5L, 7L, "trabaja")))

   // Build the initial Graph
   val graph = Graph(personas, relaciones)
   graph.vertices.collect.foreach(println(_))
// triplete entre personas  y los nombre
   //pepe tiene relacion con gonzalo
   //fran tiene una relacion con pepep....
   for  (triplet <- graph.triplets.collect) {
        println(s"${triplet.srcAttr._1}  tiene una relacion de  ${triplet.dstAttr._1}")
       }
// OTRO TRIPLETE con MAP
   //triplets

   val descripcionNatural: RDD[String] = graph.triplets.map(triplet =>
    triplet.srcAttr._1 + "  -- " + triplet.attr + " con" + triplet.dstAttr._1)
   descripcionNatural.collect.foreach(println(_))

// crear un subgrafo eliminar una arista


   val validGraph =graph.subgraph(epred=edge =>  edge.attr != "trabaja")

     validGraph.edges.collect.foreach(println)



 }

    }
```

EJERCICIO 4 CARGAR UN GRAFO DESDE FICHEROS TXT USUARIOS Y SEGUIDORES

```
import org.apache.spark.graphx.{Edge, Graph}
import org.apache.spark.rdd.RDD
import org.apache.spark.{SparkConf, SparkContext}
import org.apache.spark.SparkContext
import org.apache.spark.sql.SparkSession
```

```
import org.apache.spark.graphx._
import org.apache.spark.graphx.GraphLoader


object Ejemplo {
  def main(args: Array[String]): Unit = {

    val conf = new SparkConf().setAppName("primera").setMaster("local[*]")
    val sc = new SparkContext(conf)

    print("connected components")
    val grafoConexiones = GraphLoader.edgeListFile(sc, "seguidores.txt")

    //obtencion de vertices (nodos)
    val cc = grafoConexiones.connectedComponents().vertices

    //usuarios
    val usuarios = sc.textFile("usuarios.txt").map {line =>
      val fields = line.split(',')
      (fields(0).toLong, fields(1))
    }
   //JOIN
    print("uniendo")
    val ccByUsername = usuarios.join(cc).map {
      case (id, (user, cc)) => (user, cc)
    }
    print("pintando ")

    //printar los valores obtenidos del join (users / nod0os del componente
conectado)
    println(ccByUsername.collect().mkString("\n"))
    /*(justinbieber,1)
(matei_zaharia,3)
(ladygaga,1)
(BarackObama,1)
(jeresig,3)
(odersky,3)*/

  }
}
```

PAGE RANK → PageRank (PR) es un algoritmo utilizado por la Búsqueda de Google para clasificar los sitios web en los resultados de su motor de búsqueda.PageRank funciona contando el número y la calidad de los enlaces a una página para determinar una estimación aproximada de lo importante que es el sitio web.

Ejercicio 5 aplicando algoritmo PAGERANK

```
import org.apache.spark.graphx.{Edge, Graph}
import org.apache.spark.rdd.RDD
import org.apache.spark.{SparkConf, SparkContext}
import org.apache.spark.SparkContext
import org.apache.spark.sql.SparkSession
import org.apache.spark.graphx._
import org.apache.spark.graphx.GraphLoader


object Ejemplo {
  def main(args: Array[String]): Unit = {

    val conf = new SparkConf().setAppName("primera").setMaster("local[*]")
    val sc = new SparkContext(conf)

    print("connected components")
    val grafoConexiones = GraphLoader.edgeListFile(sc, "seguidores.txt")

    //usuarios
    val usuarios = sc.textFile("usuarios.txt").map {line =>
      val fields = line.split(',')
      (fields(0).toLong, fields(1))
    }

    //algoritmo page rank
    val ranks = grafoConexiones.pageRank(0.0001).vertices

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

    // Print the result
    print("pintand PAGERANK")
    println(ranksByUsername.collect().mkString("\n"))


    /*(justinbieber,0.15007622780470478)
(matei_zaharia,0.7017164142469724)
(ladygaga,1.3907556008752426)
(BarackObama,1.4596227918476916)
```

```
(jeresig,0.9998520559494657)
(odersky,1.2979769092759237)*/

  }
}
```

EJERCICIO 6. APLICAR EL ALGORITMO
TRIANGLE COUNT EJEMPLO(RED SOCIAL DONDE QUEREMOS SABER
SI TODOS INFLUYEN EN TODOS
SI TODOS ESTAN CONECTADOS CUANTOS + TRIANGULOS MAS CONEXIONES HABRÁ
GraphX implementa un algoritmo de recuento de triángulos en el TriangleCount objeto que determina **el número de triángulos que pasan a través de cada vértice, proporcionando una medida de agrupación en clústeres**. Calculamos el recuento de triángulos del conjunto de datos de redes sociales desde la sección PageRank. Tenga en cuenta que TriangleCount requiere que los bordes estén en orientación canónica (srcId <dstId) y el gráfico se particione mediante Graph.partitionBy.

```
Import org.apache.spark.graphx.{Edge, Graph}
import org.apache.spark.rdd.RDD
import org.apache.spark.{SparkConf, SparkContext}
import org.apache.spark.SparkContext
import org.apache.spark.sql.SparkSession
import org.apache.spark.graphx._
import org.apache.spark.graphx.GraphLoader
  import org.apache.spark.graphx.{GraphLoader, PartitionStrategy}



object Ejemplo {
 def main(args: Array[String]): Unit = {

  val conf = new SparkConf().setAppName("primera").setMaster("local[*]")
  val sc = new SparkContext(conf)

  // Load the edges in canonical order and partition the graph for triangle count
  val graph = GraphLoader.edgeListFile(sc, "seguidores.txt", true)
   .partitionBy(PartitionStrategy.RandomVertexCut)
  // Find the triangle count for each vertex
  val triCounts = graph.triangleCount().vertices
  // Join the triangle counts with the usernames
  val users = sc.textFile("usuarios.txt").map { line =>
   val fields = line.split(",")
   (fields(0).toLong, fields(1))
  }
  val triCountByUsername = users.join(triCounts).map { case (id, (username, tc)) =>
   (username, tc)
  }
  // Print the result
  println(triCountByUsername.collect().mkString("\n"))
 }
```

```
}

justinbieber,0)
(matei_zaharia,1)
(ladygaga,0)
(BarackObama,0)
(jeresig,1)
(odersky,1)
```

EJERCICIO 7. SUMAR POR AGREGACION USANDO PREGLE

```
import org.apache.spark.graphx.{Edge, Graph}
import org.apache.spark.rdd.RDD
import org.apache.spark.{SparkConf, SparkContext}
import org.apache.spark.SparkContext
import org.apache.spark.sql.SparkSession
import org.apache.spark.graphx._
import org.apache.spark.graphx.GraphLoader


object Ejemplo {
  def main(args: Array[String]): Unit = {

    val conf = new SparkConf().setAppName("primera").setMaster("local[*]")
    val sc = new SparkContext(conf)
    val vertices: RDD[(VertexId, Int)] =
      sc.parallelize(Array(
        (20L, 0)
        , (11L, 0)
        , (14L, 1000)
        , (24L, 550)
       // , (911L, 300)
      ))

    //note that the last value in the edge is for factor (positive or negative)
    val edges: RDD[Edge[Int]] =
      sc.parallelize(Array(
        Edge(14L, 11L, 1),
        Edge(24L, 11L, 1),
        Edge(11L, 20L, 1)
        //Edge(911L, 20L, 1)
      ))

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

```
  val result =
   dataItemGraph.pregel(0, activeDirection = EdgeDirection.Out)(
     (_, vd, msg) => msg + vd,t => Iterator((t.dstId, t.srcAttr)),(x, y) => x + y
   )

  result.vertices.collect().foreach(println)



 }
}
```