

**Laboratorio de Manejadores de Bases de Datos**

**Reporte de Práctica: Hands on GraphX**

**Francisco Ramos Omaña 157330**

**Erick Siordia Nagaya 157504**

**Carlos Andrés Reyes Evangelista 157068**

**10 de mayo de 2019**

Contenido

[Plan 3](#_Toc8415191)

[Objetivo 3](#_Toc8415192)

[Desarrollo 3](#_Toc8415193)

[Grafo con datos de prueba 3](#_Toc8415194)

[Grafo con datos de Wikipedia 12](#_Toc8415195)

[Ejecutando PageRank en Wikipedia 13](#_Toc8415196)

[Discusión y conclusiones 14](#_Toc8415197)

# Plan

El presente documento se divide en diferentes secciones. Primero se expone el objetivo científico de la práctica realizada. Posteriormente, los pasos seguidos son explicados en el desarrollo junto con imágenes de la ejecución de los comandos necesarios. Finalmente, en la sección de Discusión y conclusiones se explican los problemas que surgieron durante la práctica, las lecciones aprendidas y otros comentarios para concluir el documento.

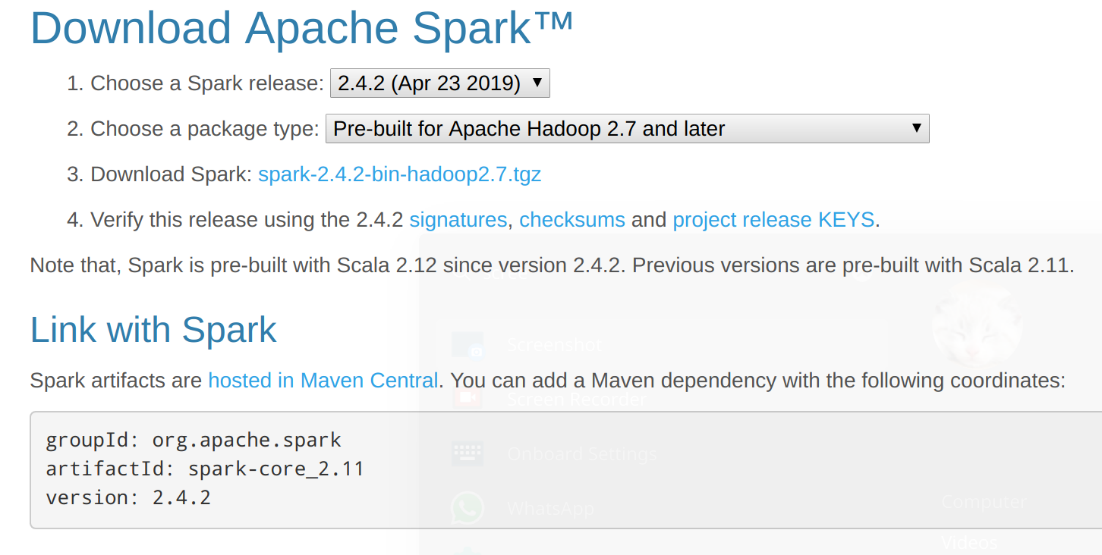
# Objetivo

# Desarrollo

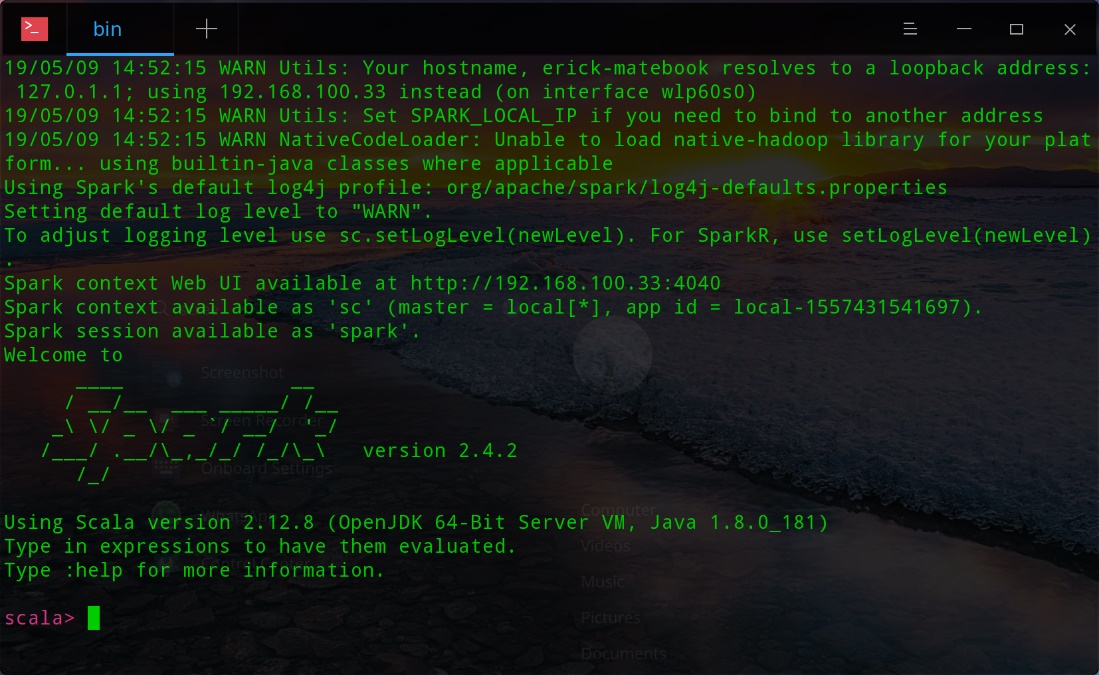
En esta sección se explicará paso a paso el proceso seguido para el desarrollo de la práctica.

## Grafo con datos de prueba

1. Primero se instaló la plataforma Apache Spark



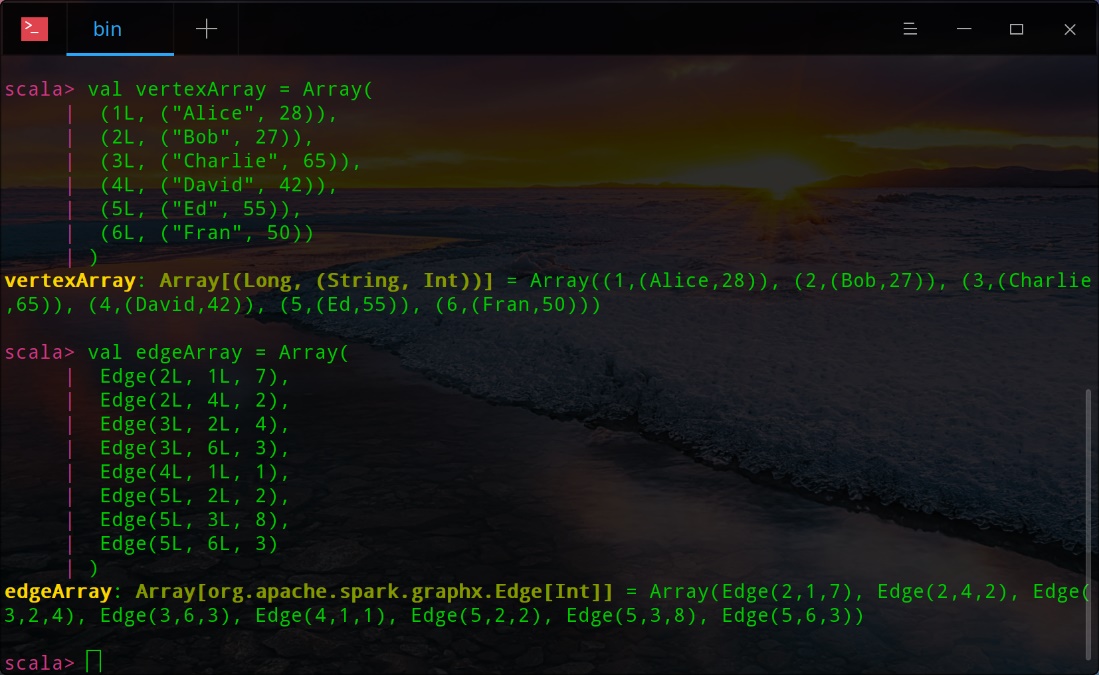
1. Se descomprimió el archivo .tar descargado y se ejecutó el archivo spark-2.4.2-bin-hadoop2.7/bin/spark-shell para obtener acceso a la terminal de Spark



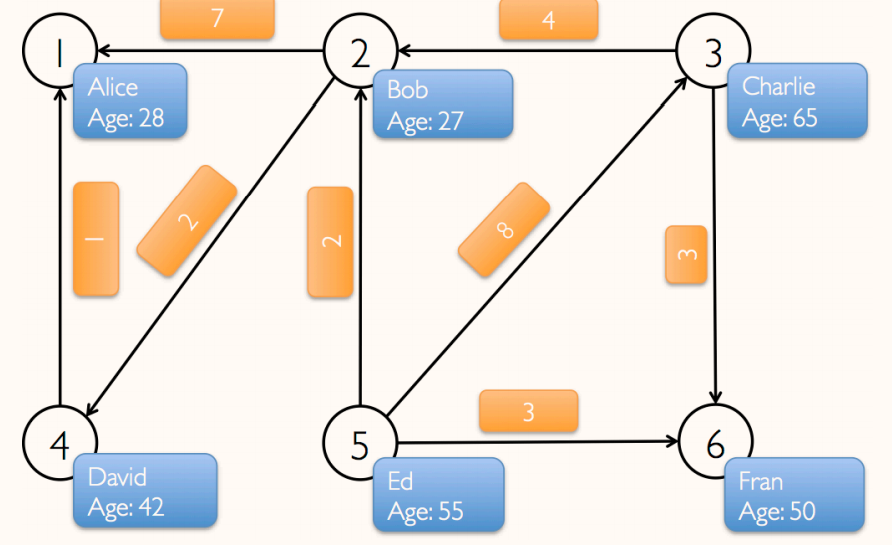
1. Se importaron los siguientes elementos del API de GraphX



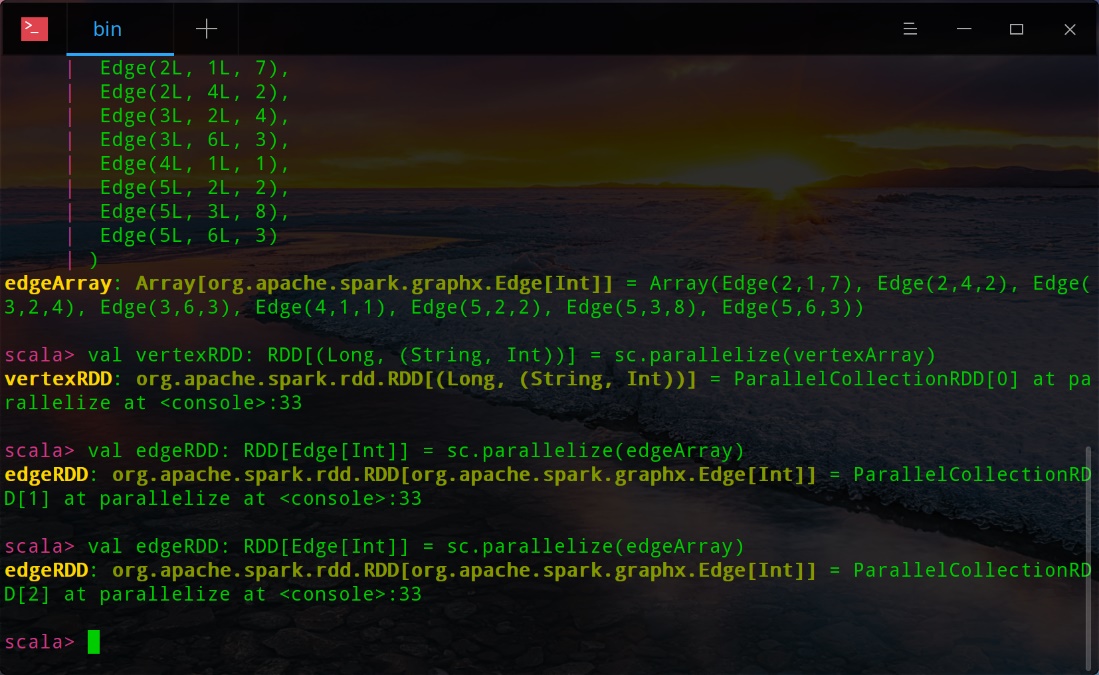
1. Se declararon un par de arreglos: uno de vértices y uno de bordes



Para el ejemplo de la práctica presente la anterior representación abstrae el concepto de una pequeña red social donde los vértices constituyen el nombre de la persona y su edad y los bordes representan el número de “me gusta” que un usuario ha dado a otro. Este ejemplo particular abstrae una representación como sigue:



1. Con ayuda de la función *sc.parallelize(),* cada uno de los arreglos convertidos fue convertido en un RDD (*Resilient Distributed Dataset*), es decir, en colecciones de datos susceptibles de ser distribuidas y capaz de tolerar fallas, errores y recuperarse de ellas –de ahí que se le conozcan como *resilientes:* aptitud que permite la adaptación y superación de situaciones adversas–.



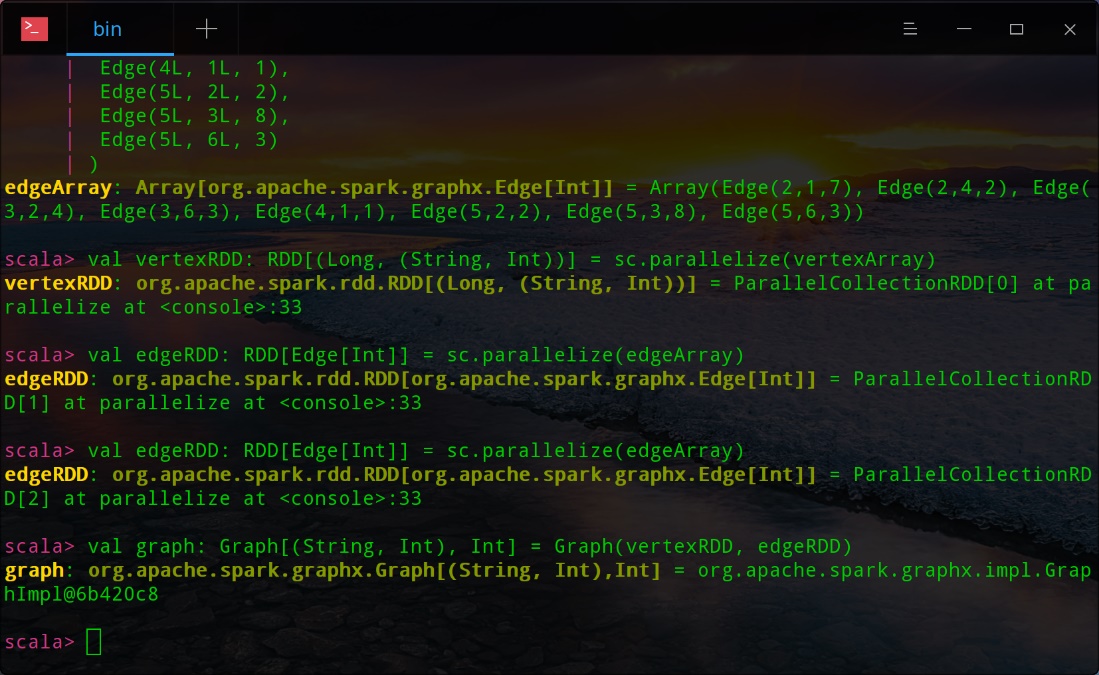
Los tipos del primer RDD (correspondiente a los vértices) [Long, (String, Int)] representan lo siguiente:

* Un número de tipo Long que representa el identificador de 64 bits único para cada vértice
* Una tupla que contiene
  + Una cadena String que mantendrá el nombre de la persona asociada al vértice
  + Un número Int que almacenará la edad de dicha persona

Para el segundo RDD se cuenta con:

* Objetos de tipo Edge que almacenan un único valor entero que representará la cantidad de *likes*

1. Se definió un nuevo grafo con el constructor por defecto de estos que toma un RDD de vértices y un RDD de bordes y construye un grafo cuyos tipos son vértices y bordes.



Cuando se declara el grafo a crear se le asignan un par de tipos: [(String, Int), Int], estos dos tipos representarán la abstracción de cada vértice y cada borde, de este modo cada vértice en el grafo se mantiene como una tupla de nombre-edad para cada usuario y cada borde es un número que representa la cantidad de “me gusta” de un usuario a otro. Ya que el grafo ha sido creado, podemos operar sobre él.

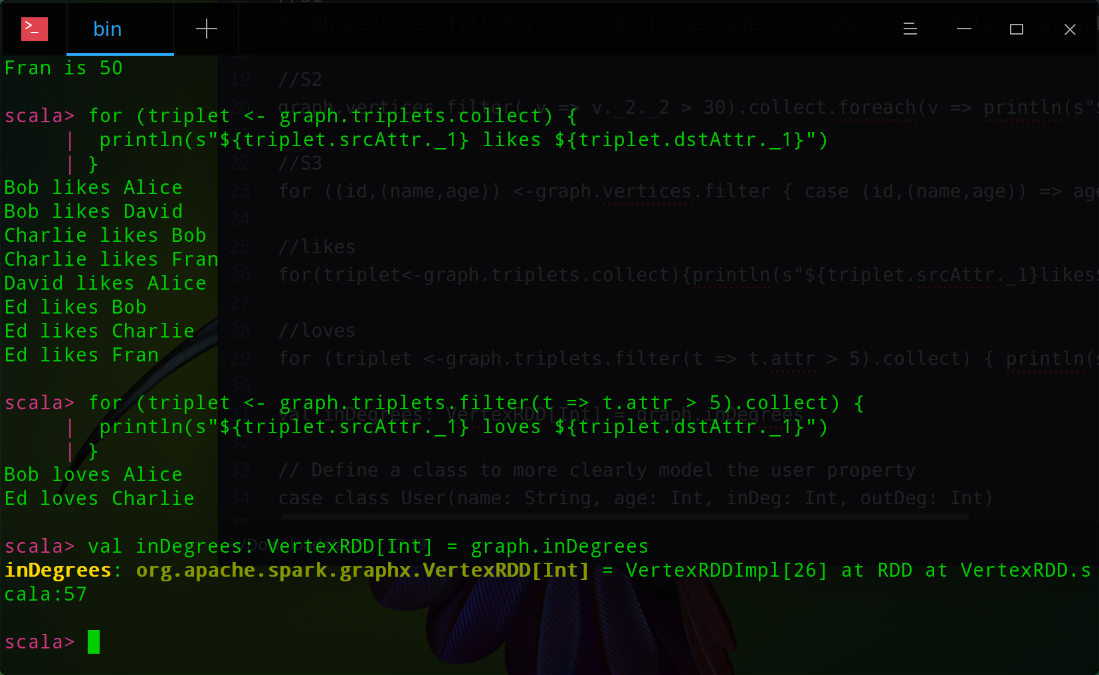
1. A continuación, una demostración acerca de cómo obtener los vértices y nodos de un grafo mediante el acceso a sus miembros **vertices** y **edges**

Para el primer comando se accede a los vértices del grafo y se filtran –por medio de la función lambda (=> age > 30)– para obtener una colección de vértices con edades superiores a 30 años, posteriormente para cada uno de esos vértices se imprime un mensaje visible.

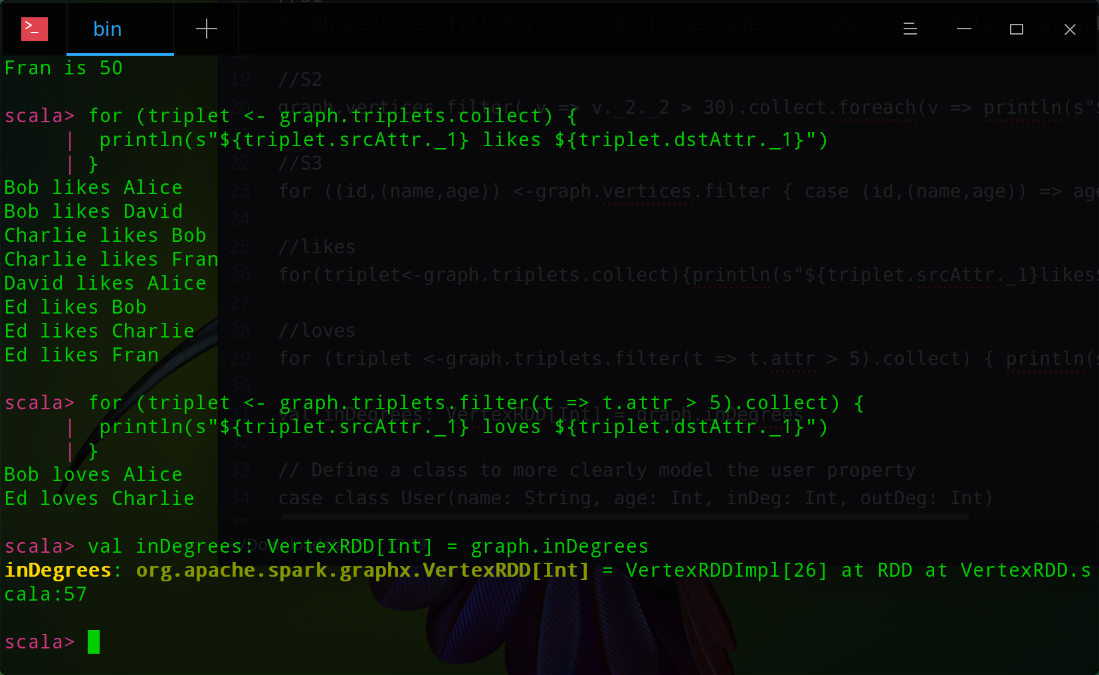
El segundo comando hace uso de la clase Triplet que representa la relación entre un borde y los dos vértices que une, mismos que son accedidos mediante las propiedades **srcAttr** y **dstAttr** que simbolizan el vértice de donde proviene y el destino, respectivamente. Este ejemplo concreto muestra quién le ha dado “me gusta” a quién.



1. El siguiente comando utiliza el mismo principio que el anterior pero además aplica un filtro para desplegar únicamente las triadas cuya cantidad de *likes* es superior a 5. La función lambda (t => t.attr > 5) indica que dado un argumento **t** (que, al iterar, tomará el valor de un Triplet) si su atributo attr es mayor que 5, ese t debería ser considerado en la lista resultante



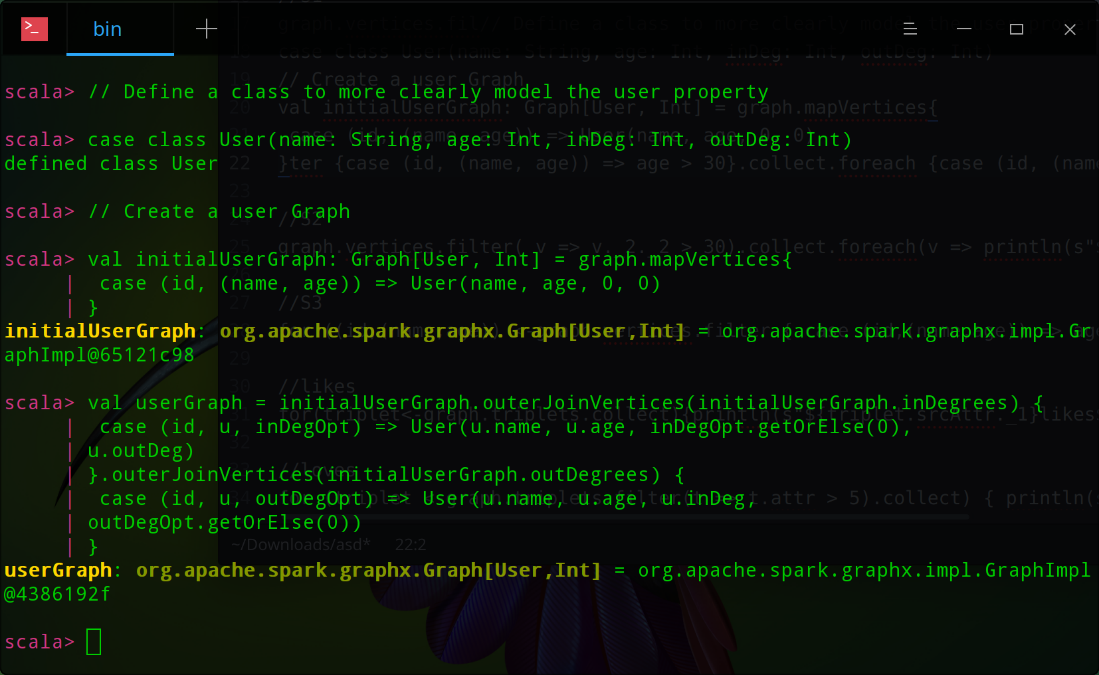
1. Posteriormente, se es calculado el grado de entrada de cada uno de los vértices mediante la utilización del miembro **inDegrees** de Graph



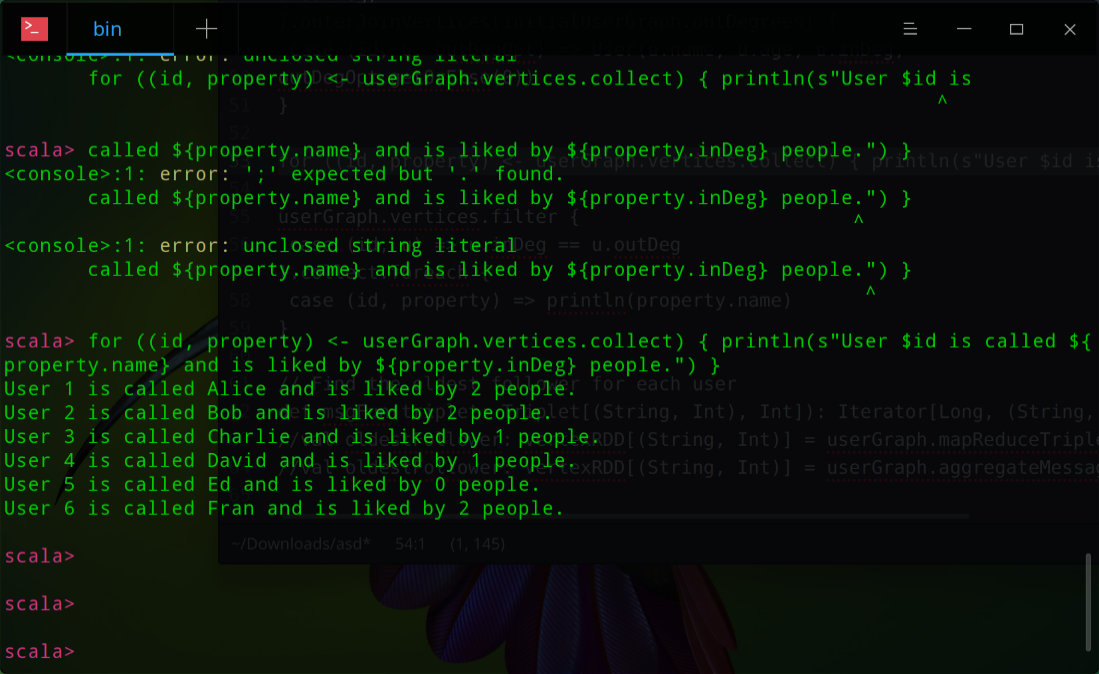
1. Instrucciones para crear una clase User para poder manipular más facilmente los datos del grafo, y para crear un grafo con esta clase. Cada User poseerá un nombre, una edad, un grado de entrada y un grado de salida.



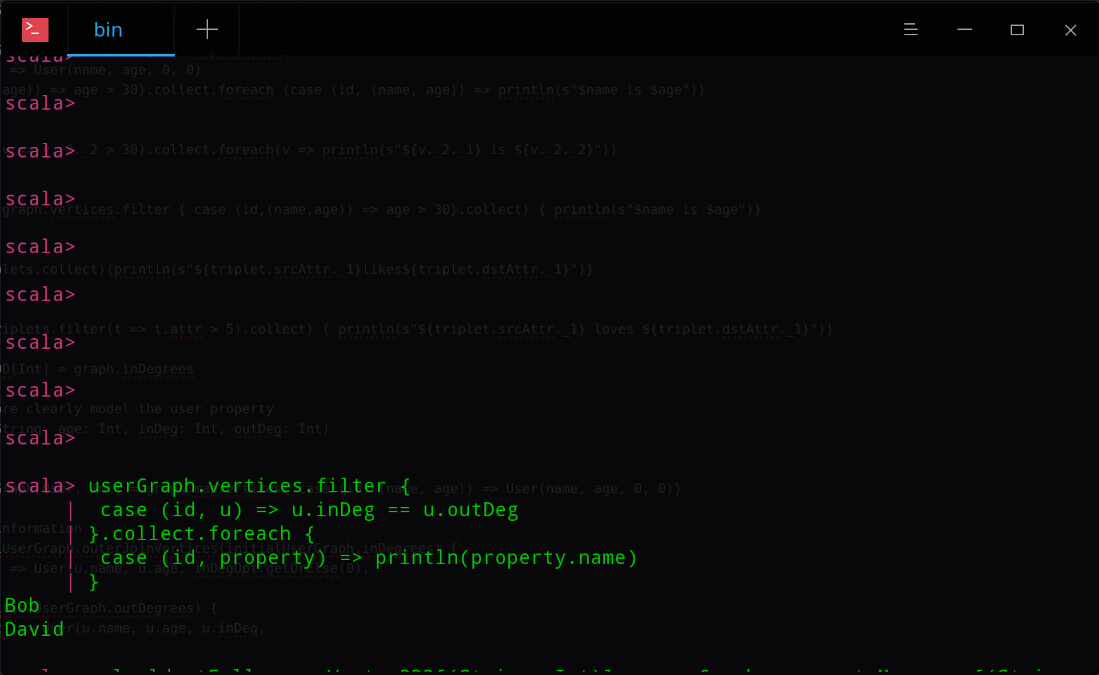
1. Instrucciones para asignar los valores de grado de entrada y salida de cada vértice del grafo a partir de los objetos de clase User.



1. Uso de la propiedad **inDegree** para conocer los datos de cada usuario y cuántas personas han dado “me gusta” a este usuario en concreto.

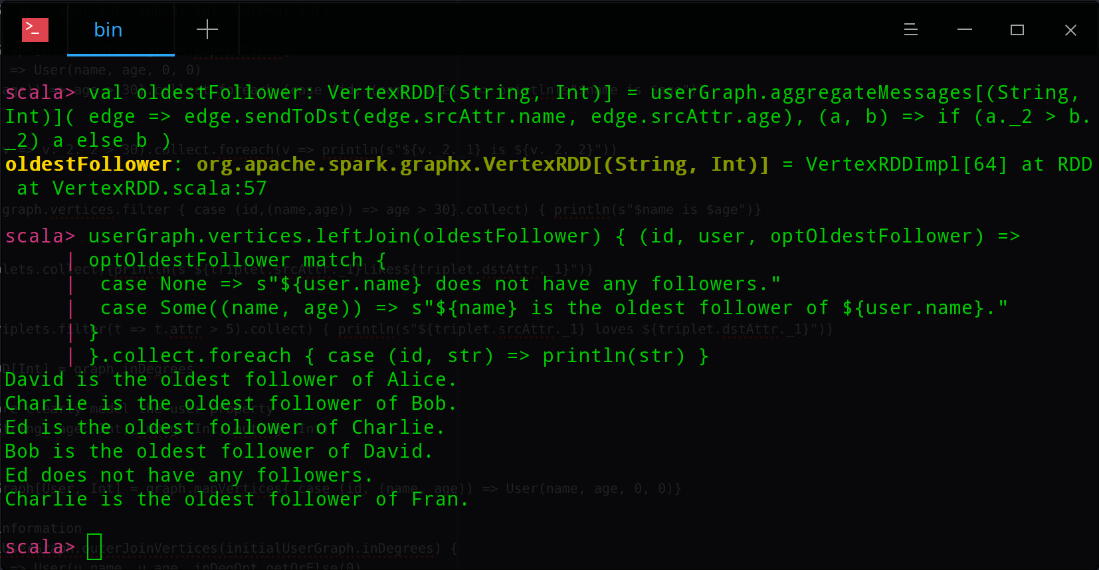


1. Comando para imprimir el nombre de las personas que tienen el mismo grado de entrada que de salida, en este contexto, que han dado la misma cantidad de *likes* que han recibido.

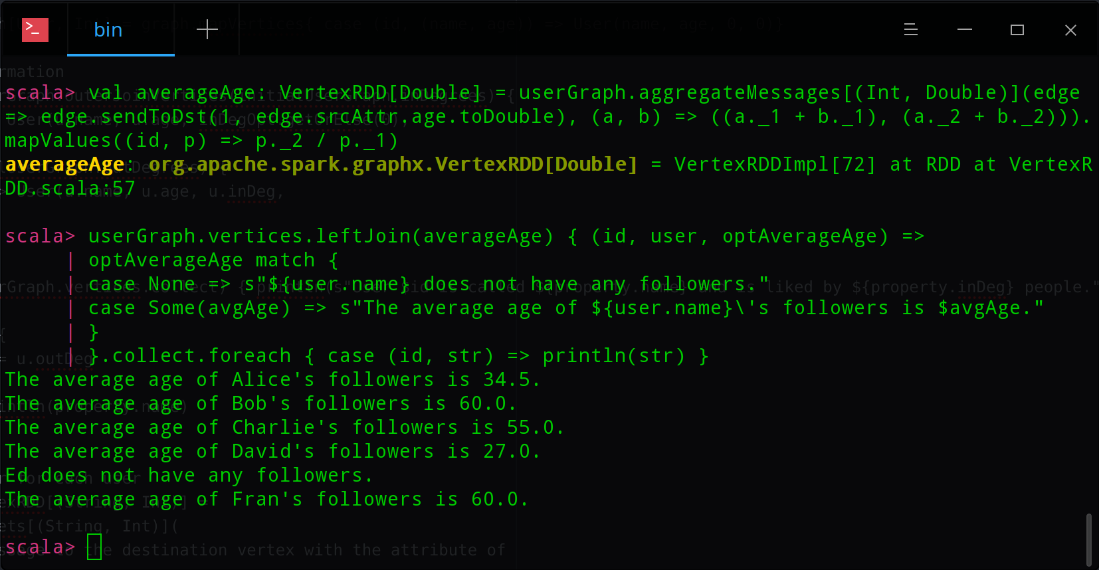


1. Instrucciones para hacer una operación de *agregación de vecindad*, donde cada vértice manda su nombre y edad a los demás vértices a los que está dirigido. Al tener todos los nodos la información de sus seguidores, podemos obtener el nombre del seguidor más viejo de cada persona en nuestro grafo.

Nótese que la implementación es diferente a la sugerida en el tutorial debido a que la función *mapReduceTriplets* es ahora obsoleta y ha sido sustituida por *aggregateMessages*. De igual forma, se utilizó la función *sendToDst* para replicar el funcionamiento del comando sugerido.

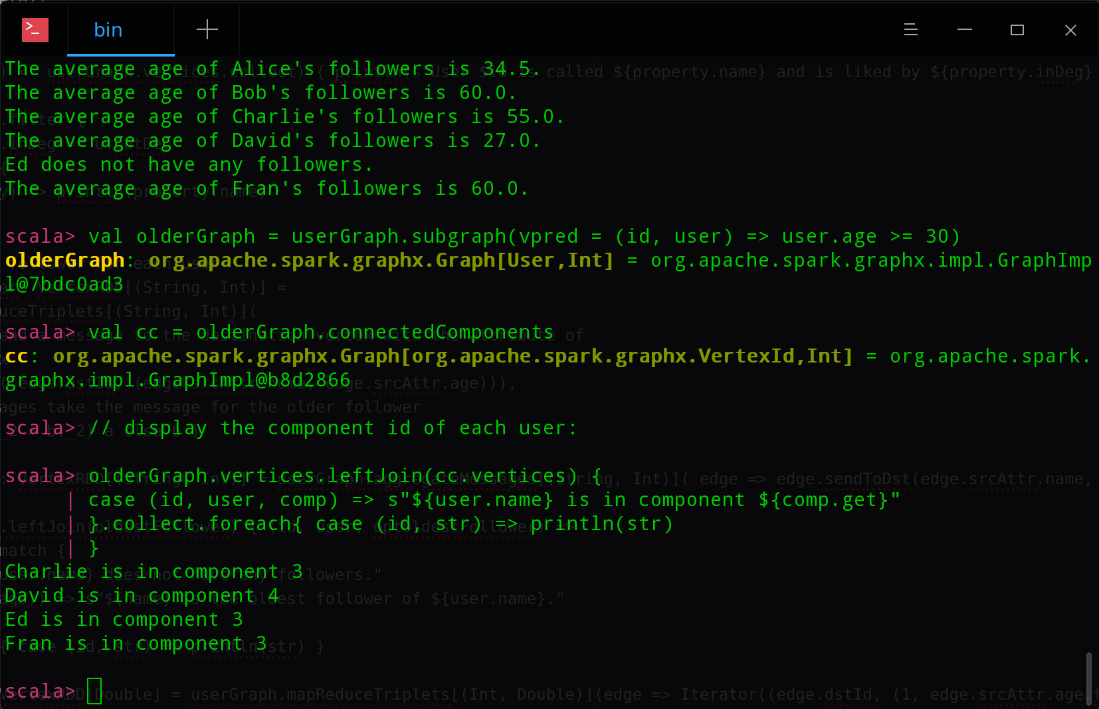


1. Bajo la misma filosofía se efectuó el siguiente comando para obtener el promedio de edad de los seguidores de las personas dentro del grafo.



1. Con el fin de realizar estudios concretos sobre conjuntos de datos puntuales ubicados en colecciones más generales, es conveniente realizar subgrafos. En esta plataforma los subgrafos son generados a partir del filtro de vértices y bordes que cumplen, cada uno, con sus condicionales respectivas. De este subgrafo creado es posible recuperar subconjuntos llamados componentes conseguidos a partir de la etiquetación de los vértices con el ID de vértice más bajo.

Este ejemplo en particular filtra para mantener únicamente usuarios de más de 30 años. David es el único que se encuentra en un componente distinto porque todos sus usuarios relacionados son menores de 30 años.

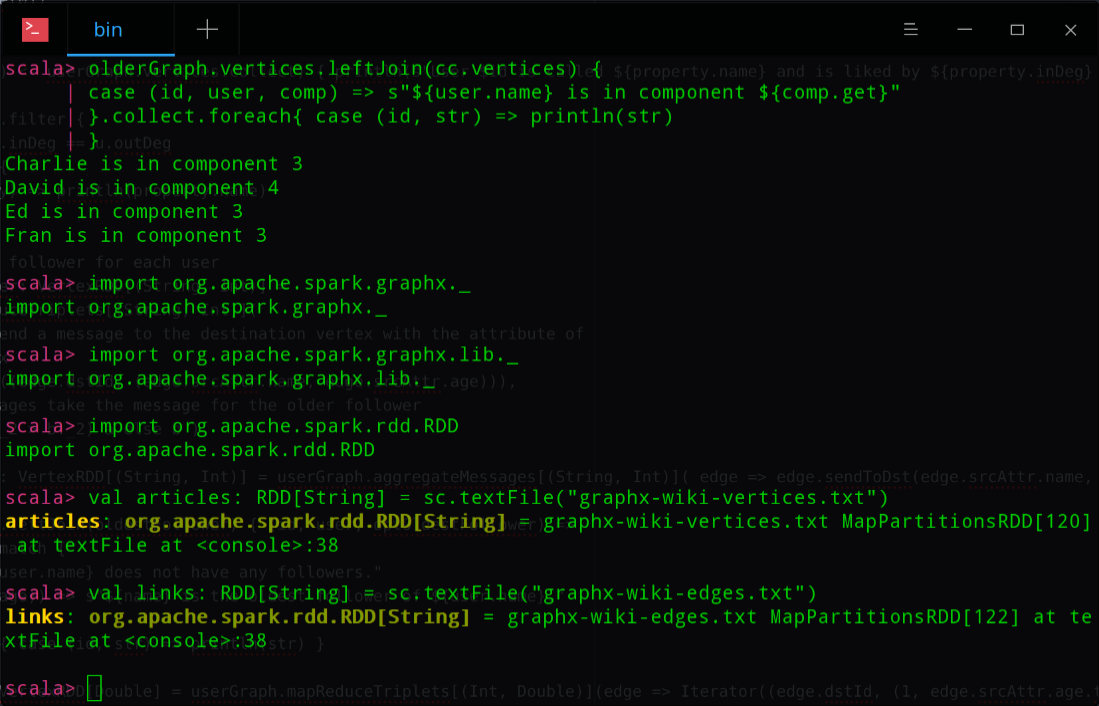


## Grafo con datos de Wikipedia

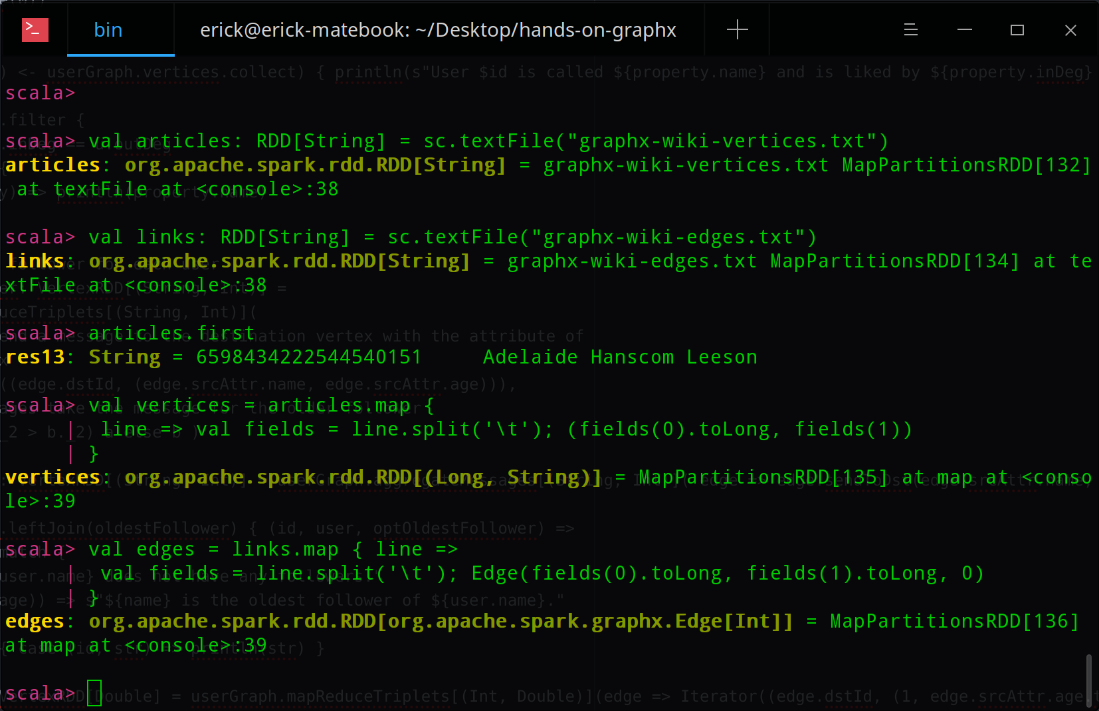
Esta parte de la práctica pretende utilizar los conceptos básicos introducidos en la sección anterior para construir un conducto analítico real. Para conseguir este objetivo se hizo uso de una serie de archivos previamente procesados con información necesaria para crear un grafo que relacione enlaces de artículos de Wikipedia con la palabra “Berkeley” en el título contra todos los artículos vinculados con cada uno de ellos.

Estos archivos contienen la información necesaria para generar los vértices, en este proyecto representados por un ID y su título, y los bordes, con información estructurada de manera: fuente-destino.

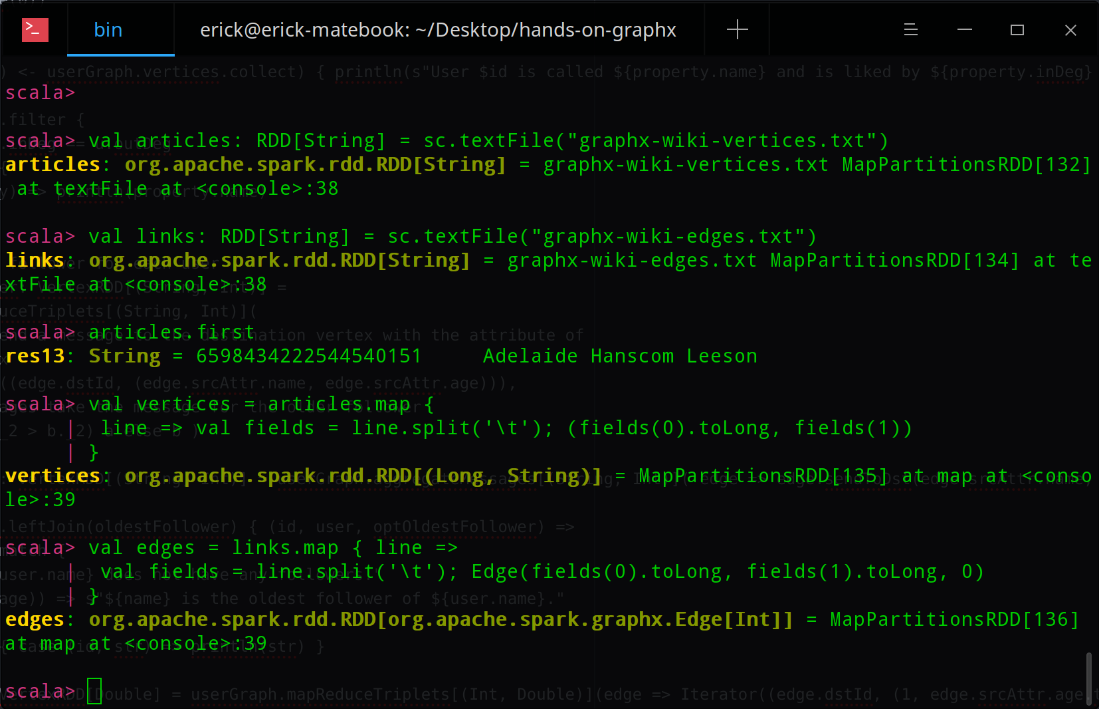
1. Con base en los archivos de texto provistos, se cargan los elementos necesarios para construir RDDs que representen los vértices y bordes, respectivamente



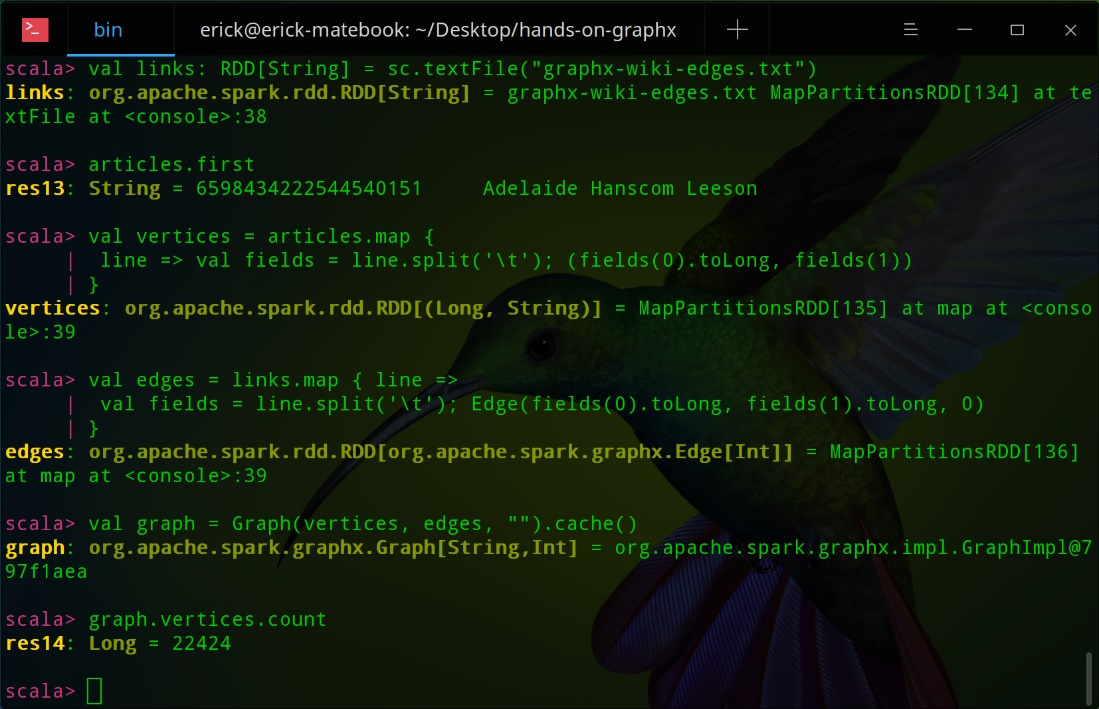
1. Se solicita al sistema el nombre del primer artículo para corroborar que todo está en orden.



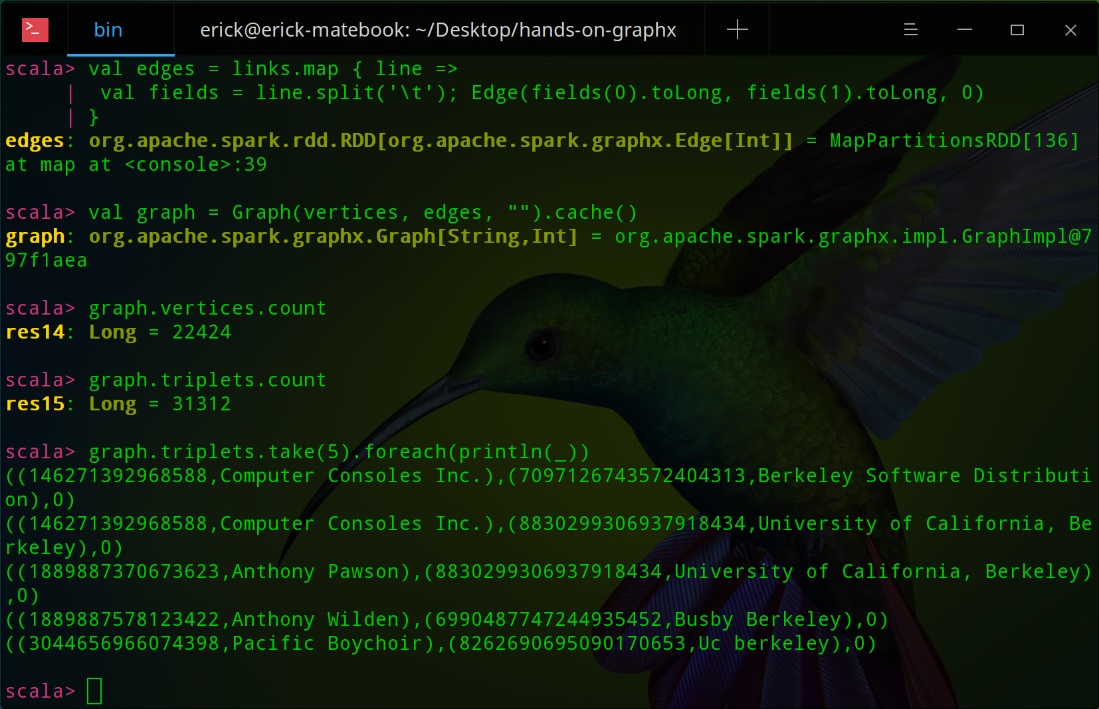
1. A partir de los RDD creados en 1) son generadas variables que contengan:
   1. Para vértices: pares que incluyen un ID y un nombre de artículo
   2. Para bordes: tuplas que conecten un ID de vértice con otro



1. Creación y cuantificación del grafo y su número de artículos. Se especificó una cadena vacía como parámetro extra para tomar en cuenta los artículos que hagan referencia a un enlace roto.

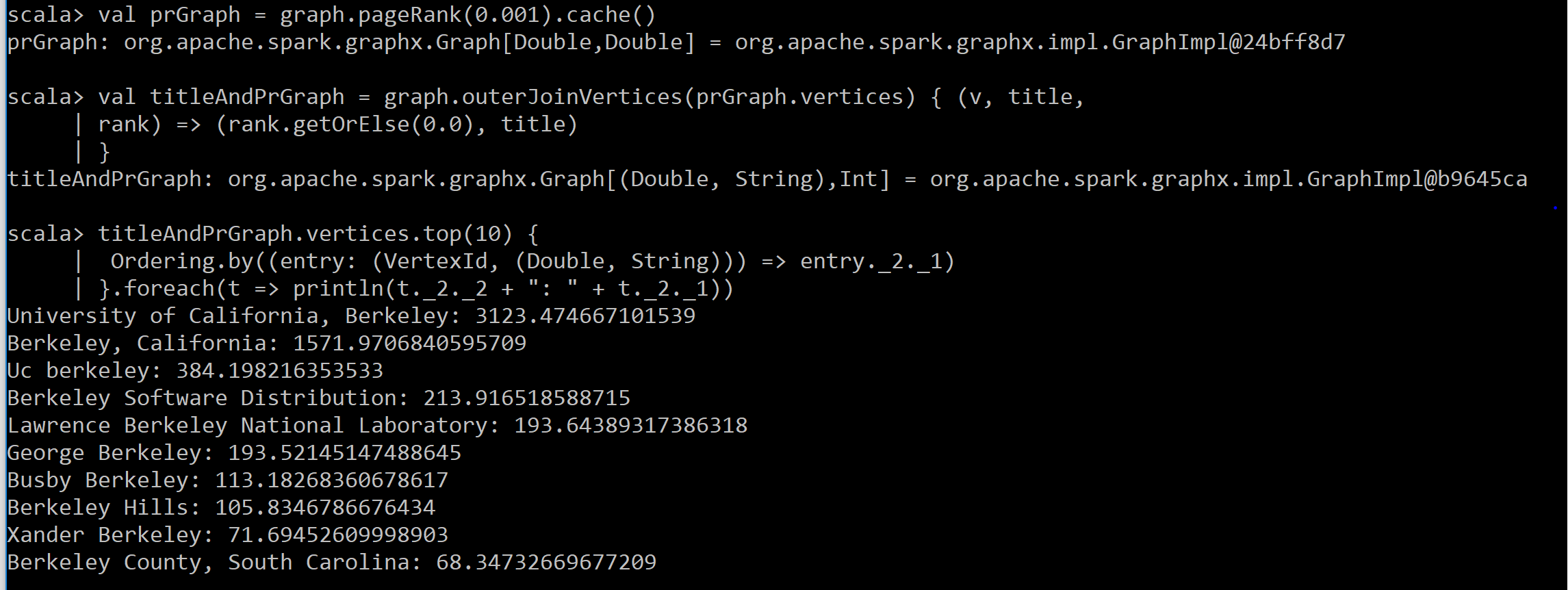


1. Cuantificación del número de tripletes en el grafo y visualización de los primeros



## Ejecutando PageRank en Wikipedia

1. Obtenemos un gráfico cuyos atributos de vértice son los valores de PageRank de cada página con una tolerancia de error de 0.001 (permite saber cuándo los rangos han convergido). Después se realizó la unión de los vértices de *prGraph* con los vértices en el gráfico que tienen la información sobre la asignación del vértice al título del artículo. Finalmente, se hallaron los diez vértices más importantes, es decir, los que tienen un rango de página más alto y se desplegaron los títulos de cada uno.



# Discusión y conclusiones

Durante el desarrollo de esta práctica pudimos aprender sobre GraphX, un sistema computacional basado en grafos en el marco de sistema de datos paralelos. Una de las ventajas de GraphX es que usa computación en memoria y tolerancia a fallos, al aprovechar las mejoras de los sistemas de flujo de datos. En esta práctica aprendimos a usar las herramientas de GraphX como el algoritmo PageRank que ayuda a determinar la “importancia” de un nodo del grafo, dependiendo del número de conexiones que tiene con otros nodos. También tuvimos que modificar una de las instrucciones de la práctica debido a que usaba el operador **mapReduceTriplets** que no es válido es las nuevas versiones, en su lugar utilizamos el operador **aggregateMessages** que expone los campos de los triplets y también funciona para enviar mensajes tanto como al vértice de origen como al de destino.

Además, por supuesto, obtuvimos nuestra primera experiencia cercana a la computación en paralelo de la mano del cuadro de trabajo Spark que, basado en la arquitectura MapReduce de Hadoop, ofrece interfaces para la programación de clústeres completos orientado a la velocidad y tolerante a fallos. A pesar de que consideramos que una práctica más exhaustiva e ilustrativa como la aplicación de estos operadores sobre un clúster pequeño real construido por nosotros habría esclarecido todos estos conceptos de una manera más certera, aun la realización de esta práctica fue útil en demasía ya que al menos ahora dominamos los conceptos base que constituyen este tipo de sistemas que realizan actividades sobre cantidades abisales de datos y la forma conceptual en que son llevadas a cabo.