UNIVERSIDAD COMPLUTENSE DE MADRID

ntic master School

Spark GraphFrames

Dr. Pablo J. Villacorta

Mayo de 2021

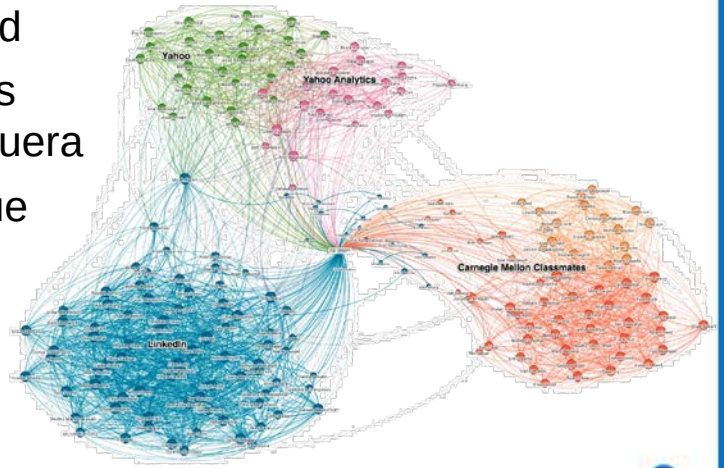
Agenda

- Introducción: grafos en la vida real
- Conceptos sobre grafos
- Análisis de grafos con Spark
  - GraphX vs GraphFrames
  - Algoritmos sobre grafos
    - Caminos más cortos
    - Componentes conexas
    - PageRank

1 Grafos en la vida real

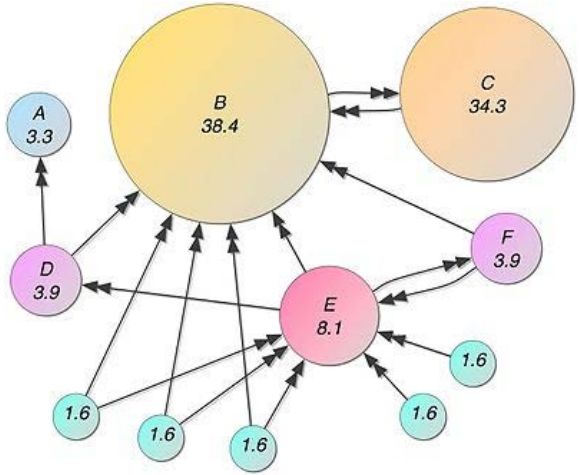
Grafos en la vida real

- Un grafo modela relaciones entre objetos en una estructura con **vértices** (nodos) y arcos (**aristas**)
- Los vértices son objetos. Los arcos conectan dos vértices si existe relación entre ellos
- Los nodos y los arcos pueden representar cualquier concepto. Ambos pueden tener propiedades asociadas
- Ejemplo perfecto: **red social**
  - Vértices: personas. Aristas: relación de amistad
  - Detección de comunidades: muchas relaciones entre sus miembros, pocas con miembros de fuera
  - Detección de personas clave (“influencers”) que participan en muchos caminos entre otras dos personas



Grafos en la vida real

- Ejemplo típico: sitios web en internet
  - Vértices: páginas web. Aristas: si un sitio web contiene un link que lleva a otro
  - El algoritmo PageRank de Google mide la importancia de un sitio como un problema de grafos, en función de cómo está relacionado con los demás, y qué importancia tienen aquellos sitios con los que está relacionado (que le enlazan)



Grafos en la vida real

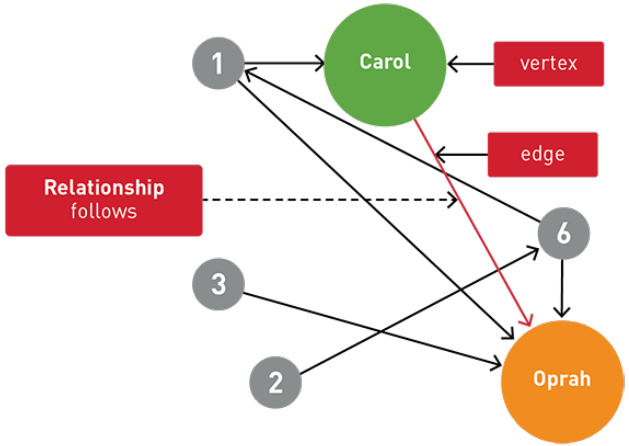
- Ejemplo interesante: identificación (tracking) de dispositivos navegando por Internet
- Vértices: dirección IP, o bien un dispositivo, o bien una cookie
  - Cookie: fichero creado por la web en el ordenador del usuario al entrar la primera vez, para identificar visitas recurrentes
- Arcos: sesiones (cuando un usuario empieza a navegar por nuestra web)
- El usuario utiliza cierto dispositivo, con cierta IP y con una cookie, que podía existir ya en su dispositivo, o no (si es la primera vez)
- Aplicaciones: la misma cookie puede usarse en varias sesiones con distinta IP (si nuestro proveedor de internet ha cambiado la IP, o nos hemos cambiado de compañía)
- Conjunto de vértices interconectados: **misma persona**

2 Conceptos sobre grafos

Concepto de grafo

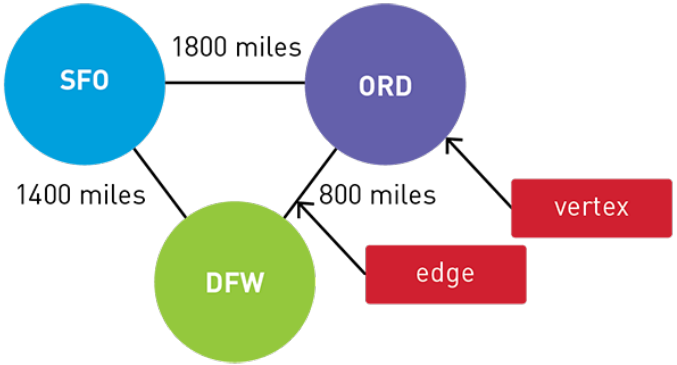
- Un grafo es una dupla G = (V, E) siendo V el conjunto de vértices y E el de aristas
- Cada vértice puede representar cualquier concepto. Incluso en un mismo grafo pueden convivir vértices que representen cosas distintas (ejemplo de cookies e IPs)
- Las aristas pueden ser dirigidas o no dirigidas (y así se denomina al grafo)
- Las aristas pueden tener pesos asociados (ej: distancia, o cualquier otra métrica del grado de intensidad de la conexión)

- Grado de entrada de un vértice: número de aristas que llegan al vértice.
- Grado de salida: número de aristas que parten de él
- Grado de un vértice: suma de ambos grados



Concepto de grafo

- Dataset de vuelos:
  - Vértices: aeropuertos
  - Aristas: existe un vuelo directo desde un aeropuerto a otro
  - Peso de cada arista: distancia entre los aeropuertos
  - Podría ser también el número de vuelos diarios entre ellos, o cualquier otra métrica



3 Grafos con Spark (GraphFrames)

Grafos con Spark: GraphX vs GraphFrames

- **GraphX** es el paquete para procesamiento y algoritmos sobre grafos con RDDs
  - Forma parte de Spark pero su API es de bajo nivel, incómoda de usar
- **GraphFrames**: evolución de GraphX con DataFrames. Será lo que veremos aquí
  - Es un paquete separado que **no forma parte de Spark** propiamente, por lo que hay que instalarlo expresamente.
  - Muy extendido. Se supone que iba a pasar a formar parte de Spark estándar pero llevan casi dos años sin fusionarlo.
  - Instalación:
    - Primero: `pip3 install graphframes` (wrapper en python)
    - Segundo: descargar fichero JAR (código Scala que es invocado desde python y que contiene realmente las implementaciones de los algoritmos en Spark)
    - Está disponible como un *Spark package* en el repo oficial de paquetes añadidos de Spark:

Grafos con GraphFrames

- GraphFrames: un grafo consiste en dos DataFrames de Spark
  - DataFrame de vértices: debe tener una columna llamada **id**
  - DataFrame de aristas: debe tener columnas llamadas **src** y **dst** con los IDs de los vértices que conecten.

```
from graphframes import GraphFrame
verticesDF = flightsDF.select(F.col("Origin").alias("id"))\
                      .distinct().cache()

edgesDF = flightsDF.withColumnRenamed("Origin", "src")\
                   .withColumnRenamed("Dest", "dst")\
                   .select("src", "dst", "ArrDelay", "Distance")\
                   .cache()

graph = GraphFrame(verticesDF, edgesDF)
```

Grafos con GraphFrames

- Una vez construido el grafo podemos aplicar algoritmos
  - Algunos algoritmos devuelven un DataFrame y otros, un grafo completamente nuevo.

- Grados de entrada y salida: el grafo calcula tres DataFrames:
  - inDegrees por vértice
  - outDegrees por vértice
  - degrees por vértice (suma de los grados de entrada y de salida)

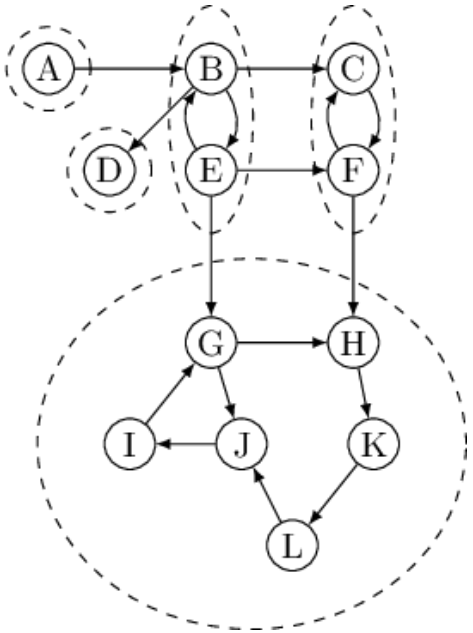
```
graph.inDegrees.orderBy(F.col("inDegree").desc()).show(3)
graph.outDegrees.orderBy(F.col("outDegree").desc()).show(3)
graph.degrees.orderBy(F.col("degree").desc()).show(3)
```

Grafos con GraphFrames

- Una vez construido el grafo podemos aplicar algoritmos:
- **Componentes conexas**: sub-grafos que están **aislados**

  - Grafos no dirigidos: dos vértices cualesquiera están conectados y no es posible viajar fuera de ese subgrafo

  - Grafos dirigidos: componentes fuertemente conexas: grupo de vértices tales que desde cualquier vértice se puede llegar a cualquier otro del grupo

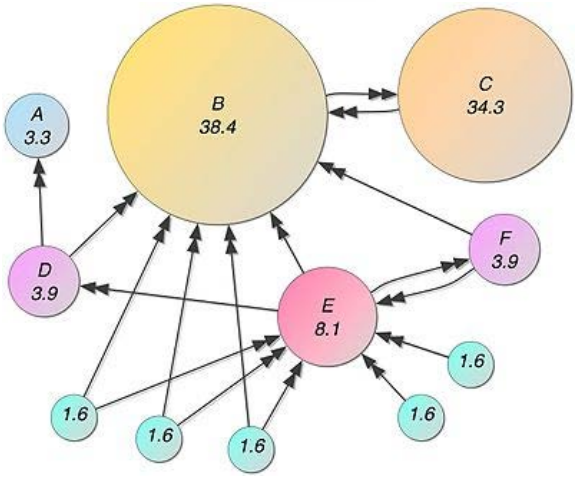


- Grupos de aeropuertos que están aislados del resto. En nuestro dataset sólo hay **una** componente conexa (formada por el grafo entero): cualquiera es alcanzable
- Este algoritmo devuelve un DataFrame

```
connCompResultDF = graph.connectedComponents()
connCompResultDF.show(100)
```

Grafos con GraphFrames

- Una vez construido el grafo podemos aplicar algoritmos:
- Algoritmo **PageRank**: algoritmo iterativo que utiliza el grado de entrada y de salida de los vértices para determinar la importancia de cada vértice.
- Es pesado desde el punto de vista computacional
- Devuelve un nuevo grafo en el que el DF de vértices tiene una nueva columna llamada **`pagerank`**




```
ranksGraph = graph.pageRank(resetProbability=0.15,
                            maxIter=10)

ranksGraph.vertices.orderBy(F.col("pagerank").desc())\
                   .show()
```

Grafos con GraphFrames

- Una vez construido el grafo podemos aplicar algoritmos:
- Algoritmo **Breadth-first search** para calcular el camino más corto entre dos conjuntos de vértices, definido como el más corto en **número de aristas**
- Los conjuntos de partida y llegada los define una expresión booleana
- Devuelve un DataFrame como resultado, con una columna que contiene la sucesión de vértices que hay que recorrer para llegar de uno a otro.

```
paths = graph.bfs(fromExpr = "id = 'LAX'", toExpr= "id = 'SFO'")

paths.show()
```

- GraphFrames no implementa algoritmos de caminos que utilicen los pesos de las artistas, como por ejemplo Dijkstra, Floyd-Warshall, etc

Grafos con GraphFrames

- Búsqueda de *motivos* (**motif finding**): búsqueda en base a **consultas estructurales**. Por ejemplo, encontrar dos aeropuertos que no estén directamente conectados pero que se pueda llegar de uno a otro a través de una conexión intermedia.

- Devuelve un DataFrame

```
res = graph.find("(a)-[]->(b); (b)-[]->(c); !(a)-[]->(c)")\
           .filter("c.id != a.id")

res.show()
```