Ecosistema Spark

Práctica 3: Grafos

José Manuel Bustos Muñoz

Objetivos

- Crear un grafo en Spark usando el API de GraphFrames
- Realizar algunas operaciones sobre ese grafo y obtener resultados.

Enunciado de la práctica

Se pide implementar código en PySpark que:

1. Cree un grafo de GraphFrames correspondiente a los ficheros de datos pasados.

2. Contenga las operaciones necesarias para contestar a las siguientes preguntas sobre el grafo creado:

- 2.1 ¿Cuántos árboles genealógicos distintos contiene el dataset proporcionado, y qué tamaño (número de personas) tienen (*definimos un árbol genealógico como el subgrafo que contiene iodos sus nodos conectados y no tiene ninguna conexión a otros nodos*)
- 2.2 ¿Cuántos ejemplos hay en el dataset de 3 personas de generaciones sucesivas (padre o madre, hijo o hija, nieto o nieta) en las que cada una tiene un país de nacimiento distinto (es decir, 3 países de nacimiento distintos)? Nota: no cuente para este cálculo las personas con país de nacimiento desconocido.
- 2.3 Encuentre la persona del dataset que más descendencia directa ha tenido (i.e. mayor número de hijos+hijas). ¿En qué año nació? ¿Qué nacionalidad tiene?. ¿Cuántos hijos+hijas tuvo?.

1. Primero se cargan los datos pasados, tanto de los vértices como de los enlaces entre ellos.

```
# cargar los datos de ambos ficheros
vertices = spark.read.csv('./p3_spark/spark-grafos-practica/vertices-subset.csv', inferSchema=True, header=True,\
                          nullValue='\\N')
edges = spark.read.csv('./p3_spark/spark-grafos-practica/edges-subset.csv', inferSchema=True, header=True,\
                          nullValue='\\N')
```

Una vez cargados se importa la librería de graphframes, y se crea el grado pasándole los datos cargados anteriormente.

```
# importamos graphframes
import graphframes as gf
```

```
# creamos el grafo a partir de los vertices y edges cargados
g = gf.GraphFrame(vertices, edges)
```

```
# comprobamos que el grafo creado tiene los mismos datos que los ficheros cargados
print("Edges: ", g.edges.count(), "\tVertices: ", g.vertices.count())
```

```
Edges:  7308    Vertices:  4527
```

Una vez creado el grafo y comprobado que se han cargado los datos en el mismo, se puede representar.

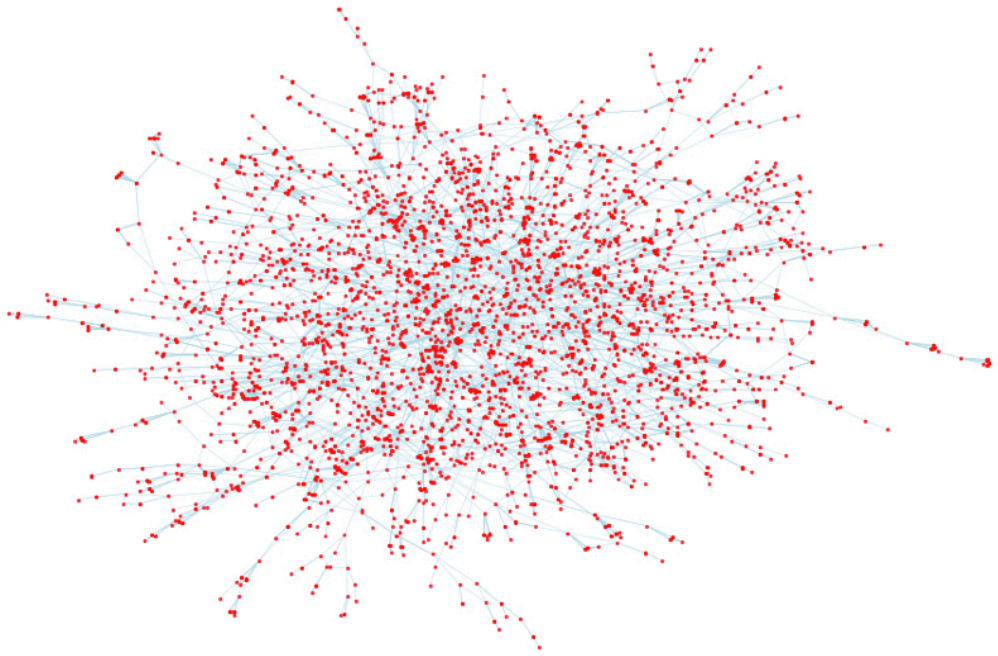
```
# Representación gráfica del grafo
pd_edges = g.edges.toPandas()
pd_nodes = g.vertices.toPandas()
```

```
import networkx as nx
gnx = nx.from_pandas_dataframe(pd_edges,'src','dst')
for n in pd_nodes.itertuples(index=False):
    gnx.add_node(n.id, **n._asdict())
```

```
import matplotlib.pyplot as plt
%matplotlib inline

fig, ax = plt.subplots(1, 1, figsize=(18, 12))
nx.draw_networkx(gnx, ax=ax, with_labels=False,
                 node_size=5, width=.5, alpha=0.75, edge_color='#add8e6')
ax.set_axis_off()
```

Representación gráfica del grafo creado con los datos de la práctica.



2.

2.1 Se utiliza connectedComponents() para obtener los distintos componentes conectados que tiene el grafo.

```
# objeto de componentes conectados en el grafo
sc.setCheckpointDir('./tmp/cc')
cc = g.connectedComponents()
```

Agrupando por componente y haciendo un recuento obtenemos los distintos componentes y el número de elementos de cada uno.

```
# agrupamos por componente y calculamos el recuento de registros de cada uno
cc_ord.groupBy('component').count().orderBy('count', ascending=False).toPandas()
```

	component	count
0	55143	1658
1	24553	1488
2	225398	1381

Se obtienen 3 componentes dentro del grafo, con 1658 personas, 1488 y 1381 respectivamente en cada árbol genealógico.

2.2 Partiendo del grafo inicial podemos filtrar para eliminar los elementos que no tengan especificado el país de nacimiento. Una vez realizado dicho filtro, utilizando "find" podemos buscar las rutas de padre-hijo-nieto, y por último sobre las rutas o caminos encontrados volver a filtrar para quedarnos con aquellos donde cada elemento tiene un país de nacimiento diferente.

```
# Partiendo del grafo, filtramos los vértices donde el atributo del pais de nacimiento sea igual a *
gvertices = g.vertices.filter('birth_location_country != "*"')
```

```
# generamos un segundo grafo pasándole los vértices que se obtuvieron antes y tengan el pais de nacimiento con un
# valor correcto
g2 = gf.GraphFrame(gvertices, g.edges)
```

```
# sobre el grafo generado anteriormente buscamos los caminos a-->b-->c
ruta_3_gen = g2.find("(a)-[e1]->(b); (b)-[e2]->(c)")
ruta_3_gen.count()
```

```
5770
```

```
# filtramos los caminos obtenidos para quedarnos con aquellos donde el pais de nacimiento es diferente para el nodo a,
# el nodo b, y el nodo c.
ruta_3_gen.filter('a.birth_location_country!=b.birth_location_country \
                   AND b.birth_location_country!=c.birth_location_country \
                   AND c.birth_location_country!=a.birth_location_country').toPandas()
```

Se obtienen 42 ejemplos de generaciones padre-hijo-nieto donde cada persona tiene un país de nacimiento diferente.

2.3 Utilizamos "outdegrees" para obtener el grado de salida de cada nodo, los enlaces que salen de él, que en este caso serían los descendientes directos de una persona. Lo unimos a los vértices o nodos, creándose una columna para cada registro que sería el grado de salida. Así podría ordenarse de mayor a menor y arriba tendríamos la persona o personas con mayor número de hijos/as.

```
# Otra forma de hacerlo, añadiendo al df de vértices/personas un atributo con los grados de salida de cada id
df_with_outdegrees = g.outDegrees.join(vertices, 'id').toPandas()
df_with_outdegrees
```

```
# ordenamos de forma descendente por el atributo outDegree mostrando sólo los 5 primeros, donde pueden verse de nuevo
# las dos personas con mayor número de descendientes.
df_with_outdegrees.sort_values('outDegree',ascending=False).head(2)
```

	id	outDegree	gender	is_alive	birth_year	birth_location_city	birth_location_country	death_year	death_location_country	cause_o
301	43475617	16	female	0	1704	Isfield	England	1780	England	*
1452	43871304	16	male	0	1700	Isfield	England	1767	England	*

Las dos personas que mayor descendencia directa tuvieron, 16 hijos/as, serían un hombre y una mujer que nacieron en 1700 y 1704 respectivamente, y ambos son de nacionalidad inglesa.