Creado por:

Isabel Maniega

GraphFrames

Realizaremos la instalación de la librería de Graphframes:

In [1]:
```
# pip install graphframes
```

In [3]:
```
import pyspark
import os
```

In [4]:
```
os.environ["PYSPARK_SUBMIT_ARGS"] = "--packages graphframes:graphframes:0
```

In [5]:
```
conf = pyspark.SparkConf()
```

In [6]:
```
sc = pyspark.SparkContext(conf=conf)
print(sc._conf.getAll())
```

```
[('spark.driver.extraJavaOptions', '-XX:+IgnoreUnrecognizedVMOptions --a
dd-opens=java.base/java.lang=ALL-UNNAMED --add-opens=java.base/java.lan
g.invoke=ALL-UNNAMED --add-opens=java.base/java.lang.reflect=ALL-UNNAMED
--add-opens=java.base/java.io=ALL-UNNAMED --add-opens=java.base/java.net
=ALL-UNNAMED --add-opens=java.base/java.nio=ALL-UNNAMED --add-opens=jav
a.base/java.util=ALL-UNNAMED --add-opens=java.base/java.util.concurrent=
ALL-UNNAMED --add-opens=java.base/java.util.concurrent.atomic=ALL-UNNAME
D --add-opens=java.base/sun.nio.ch=ALL-UNNAMED --add-opens=java.base/su
n.nio.cs=ALL-UNNAMED --add-opens=java.base/sun.security.action=ALL-UNNAM
ED --add-opens=java.base/sun.util.calendar=ALL-UNNAMED --add-opens=java.
security.jgss/sun.security.krb5=ALL-UNNAMED'), ('spark.app.initial.jar.u
rls', 'spark://dc42b531111e:39853/jars/org.slf4j_slf4j-api-1.7.16.jar,sp
ark://dc42b531111e:39853/jars/graphframes_graphframes-0.8.1-spark3.0-s_
2.12.jar'), ('spark.app.submitTime', '1669719894100'), ('spark.executor.
id', 'driver'), ('spark.submit.pyFiles', '/home/jovyan/.ivy2/jars/graphf
rames_graphframes-0.8.1-spark3.0-s_2.12.jar,/home/jovyan/.ivy2/jars/org.
slf4j_slf4j-api-1.7.16.jar'), ('spark.app.name', 'pyspark-shell'), ('spa
rk.jars', 'file:///home/jovyan/.ivy2/jars/graphframes_graphframes-0.8.1-
spark3.0-s_2.12.jar,file:///home/jovyan/.ivy2/jars/org.slf4j_slf4j-api-
1.7.16.jar'), ('spark.app.id', 'local-1669719894732'), ('spark.app.start
Time', '1669719894214'), ('spark.rdd.compress', 'True'), ('spark.executo
r.extraJavaOptions', '-XX:+IgnoreUnrecognizedVMOptions --add-opens=java.
base/java.lang=ALL-UNNAMED --add-opens=java.base/java.lang.invoke=ALL-UN
NAMED --add-opens=java.base/java.lang.reflect=ALL-UNNAMED --add-opens=ja
va.base/java.io=ALL-UNNAMED --add-opens=java.base/java.net=ALL-UNNAMED -
-add-opens=java.base/java.nio=ALL-UNNAMED --add-opens=java.base/java.uti
l=ALL-UNNAMED --add-opens=java.base/java.util.concurrent=ALL-UNNAMED --a
dd-opens=java.base/java.util.concurrent.atomic=ALL-UNNAMED --add-opens=j
ava.base/sun.nio.ch=ALL-UNNAMED --add-opens=java.base/sun.nio.cs=ALL-UNN
AMED --add-opens=java.base/sun.security.action=ALL-UNNAMED --add-opens=j
ava.base/sun.util.calendar=ALL-UNNAMED --add-opens=java.security.jgss/su
n.security.krb5=ALL-UNNAMED'), ('spark.app.initial.file.urls', 'file:///
home/jovyan/.ivy2/jars/graphframes_graphframes-0.8.1-spark3.0-s_2.12.ja
r,file:///home/jovyan/.ivy2/jars/org.slf4j_slf4j-api-1.7.16.jar'), ('spa
rk.serializer.objectStreamReset', '100'), ('spark.master', 'local[*]'),
('spark.submit.deployMode', 'client'), ('spark.driver.port', '39853'),
('spark.driver.host', 'dc42b531111e'), ('spark.files', 'file:///home/jov
yan/.ivy2/jars/graphframes_graphframes-0.8.1-spark3.0-s_2.12.jar,fil
e:///home/jovyan/.ivy2/jars/org.slf4j_slf4j-api-1.7.16.jar'), ('spark.re
pl.local.jars', 'file:///home/jovyan/.ivy2/jars/graphframes_graphframes-
0.8.1-spark3.0-s_2.12.jar,file:///home/jovyan/.ivy2/jars/org.slf4j_slf4j
-api-1.7.16.jar'), ('spark.ui.showConsoleProgress', 'true')]
```

Creamos el Grafo

In [7]:
```
from pyspark import *
from pyspark.sql import *
from graphframes import *
```

In [13]:
```
spark = SparkSession.builder.appName('graphframes').getOrCreate()

v = spark.createDataFrame([
    ("A", "ANA" , 350 ),
    ("B", "BERTO", 360 ),
    ("C", "CLARA", 195 ),
    ("D", "DANIEL", 90),
    ("E", "ERICA", 90),
    ("F", "FRANCISCO", 215 ),
```

```
    ("G", "GERARDO", 30 ),
    ("H", "HERNANDO", 25 ),
    ("I", "INMA", 25 ),
    ("J", "JUAN" , 20 )
], ["id", "name", "total_points"])
v.show()
```

```
+---+---------+------------+
| id|     name|total_points|
+---+---------+------------+
|  A|      ANA|         350|
|  B|    BERTO|         360|
|  C|    CLARA|         195|
|  D|   DANIEL|          90|
|  E|    ERICA|          90|
|  F|FRANCISCO|         215|
|  G|  GERARDO|          30|
|  H| HERNANDO|          25|
|  I|     INMA|          25|
|  J|     JUAN|          20|
+---+---------+------------+
```

In [14]:
```
e = spark.createDataFrame([
    ("A", "B" , 60),
    ("B", "A", 60),
    ("A", "C", 50),
    ("D", "A", 100),
    ("A", "D", 80),
    ("C", "I", 25),
    ("C", "J", 20),
    ("B", "F", 50),
    ("F", "B", 60),
    ("F", "G" , 110),
    ("F", "H", 25),
    ("B", "E", 90),
], ["src", "dst", "relationship"])
e.show()
```

```
+---+---+------------+
|src|dst|relationship|
+---+---+------------+
|  A|  B|          60|
|  B|  A|          60|
|  A|  C|          50|
|  D|  A|         100|
|  A|  D|          80|
|  C|  I|          25|
|  C|  J|          20|
|  B|  F|          50|
|  F|  B|          60|
|  F|  G|         110|
|  F|  H|          25|
|  B|  E|          90|
+---+---+------------+
```

In [15]:
```
g = GraphFrame(v,e)
```

```
/usr/local/spark/python/pyspark/sql/dataframe.py:148: UserWarning: DataF
rame.sql_ctx is an internal property, and will be removed in future rele
ases. Use DataFrame.sparkSession instead.
  warnings.warn(
```

Atributos básicos Graph:

- Definimos cual corresponde al vértice y cual a la arista:

In [16]:
```
verticesDF = g.vertices
edgesDF = g.edges
```

In [17]:
```
verticesDF.show()
```

```
+---+---------+------------+
| id|     name|total_points|
+---+---------+------------+
|  A|      ANA|         350|
|  B|    BERTO|         360|
|  C|    CLARA|         195|
|  D|   DANIEL|          90|
|  E|    ERICA|          90|
|  F|FRANCISCO|         215|
|  G|  GERARDO|          30|
|  H| HERNANDO|          25|
|  I|     INMA|          25|
|  J|     JUAN|          20|
+---+---------+------------+
```

In [18]:
```
edgesDF.show()
```

```
+---+---+------------+
|src|dst|relationship|
+---+---+------------+
|  A|  B|          60|
|  B|  A|          60|
|  A|  C|          50|
|  D|  A|         100|
|  A|  D|          80|
|  C|  I|          25|
|  C|  J|          20|
|  B|  F|          50|
|  F|  B|          60|
|  F|  G|         110|
|  F|  H|          25|
|  B|  E|          90|
+---+---+------------+
```

- inDegrees, outDegrees y degrees:
  - **inDegrees:** son las entradas del nodo
  - **outDegrees:** son las salidas del nodo
  - **degrees:** son todas las conexiones por nodo

In [21]:
```
inDegreesDF = g.inDegrees
outDegreesDF = g.outDegrees
degreesDF = g.degrees
```

In [23]:
```
inDegreesDF.sort(['inDegree'],ascending=[0]).show() # Sort and show
```

```
+---+--------+
| id|inDegree|
+---+--------+
|  B|       2|
|  A|       2|
|  C|       1|
|  D|       1|
|  J|       1|
|  G|       1|
|  I|       1|
|  F|       1|
|  E|       1|
|  H|       1|
+---+--------+
```

In [24]:
```
outDegreesDF.sort(['outDegree'],ascending=[0]).show() # Sort and show
```

```
+---+---------+
| id|outDegree|
+---+---------+
|  F|        3|
|  B|        3|
|  A|        3|
|  C|        2|
|  D|        1|
+---+---------+
```

In [26]:
```
degreesDF.show() # Sort and show
```

```
+---+------+
| id|degree|
+---+------+
|  B|     5|
|  A|     5|
|  C|     3|
|  D|     2|
|  I|     1|
|  J|     1|
|  F|     4|
|  G|     1|
|  E|     1|
|  H|     1|
+---+------+
```

Estructura Graph

- **PageRank:** Permite calcular los pesos para cada nodo o lo que es lo mismo asignar de forma numérica la relevancia de los nodos.

In [27]:
```
PageRankResult = g.pageRank(resetProbability=0.15, tol=0.01)
PageRankResult.vertices.sort(["pageRank"],ascending=[0]).show()
```

```
+---+---------+------------+------------------+
| id|     name|total_points|          pagerank|
+---+---------+------------+------------------+
|  A|      ANA|         350|1.6755780131663474|
|  B|    BERTO|         360|1.2302848386832057|
|  D|   DANIEL|          90|  0.99894394905248|
|  C|    CLARA|         195|  0.99894394905248|
|  J|     JUAN|          20|0.9313848137403751|
|  I|     INMA|          25|0.9313848137403751|
|  E|    ERICA|          90|0.8612020763296273|
|  F|FRANCISCO|         215|0.8612020763296273|
|  G|  GERARDO|          30|0.7555377349527411|
|  H| HERNANDO|          25|0.7555377349527411|
+---+---------+------------+------------------+
```

In [29]:
```
PageRankResult.edges.show()
```

```
+---+---+------------+------------------+
|src|dst|relationship|            weight|
+---+---+------------+------------------+
|  F|  B|          60|0.3333333333333333|
|  F|  G|         110|0.3333333333333333|
|  F|  H|          25|0.3333333333333333|
|  B|  F|          50|0.3333333333333333|
|  B|  E|          90|0.3333333333333333|
|  B|  A|          60|0.3333333333333333|
|  D|  A|         100|               1.0|
|  C|  J|          20|               0.5|
|  C|  I|          25|               0.5|
|  A|  B|          60|0.3333333333333333|
|  A|  D|          80|0.3333333333333333|
|  A|  C|          50|0.3333333333333333|
+---+---+------------+------------------+
```

- **Label Propagation Algorithm (LPA):** Poder obtener la relación entre nodos obteniendo los grupos que lo forman, asignando una etiqueta a cada grupo.

In [30]:
```
result = g.labelPropagation(maxIter=5)
result.sort(["label"],ascending=[0]).show()
```

```
+---+---------+------------+-------------+
| id|     name|total_points|        label|
+---+---------+------------+-------------+
|  C|    CLARA|         195|1391569403904|
|  I|     INMA|          25| 764504178688|
|  J|     JUAN|          20| 764504178688|
|  F|FRANCISCO|         215| 420906795008|
|  A|      ANA|         350| 420906795008|
|  E|    ERICA|          90| 420906795008|
|  B|    BERTO|         360| 171798691840|
|  D|   DANIEL|          90| 171798691840|
|  G|  GERARDO|          30| 171798691840|
|  H| HERNANDO|          25| 171798691840|
+---+---------+------------+-------------+
```

- **Connected Components:** Observar cuales la conexión entre los nodos, si hay nodos aislados o no.

In [32]:
```
sc.setCheckpointDir("graphframes_cps")
result = g.connectedComponents()
result.show()
```

```
+---+---------+------------+------------+
| id|     name|total_points|   component|
+---+---------+------------+------------+
|  A|      ANA|         350|171798691840|
|  B|    BERTO|         360|171798691840|
|  C|    CLARA|         195|171798691840|
|  D|   DANIEL|          90|171798691840|
|  E|    ERICA|          90|171798691840|
|  F|FRANCISCO|         215|171798691840|
|  G|  GERARDO|          30|171798691840|
|  H| HERNANDO|          25|171798691840|
|  I|     INMA|          25|171798691840|
|  J|     JUAN|          20|171798691840|
+---+---------+------------+------------+
```

- **Strongly Connected Components:** Ver que nodos están fuertemente conectados.

In [33]:
```
result = g.stronglyConnectedComponents(maxIter=10)
result.show()
```

```
+---+---------+------------+-------------+
| id|     name|total_points|    component|
+---+---------+------------+-------------+
|  F|FRANCISCO|         215| 171798691840|
|  E|    ERICA|          90| 369367187456|
|  B|    BERTO|         360| 171798691840|
|  D|   DANIEL|          90| 171798691840|
|  C|    CLARA|         195| 764504178688|
|  J|     JUAN|          20| 807453851648|
|  A|      ANA|         350| 171798691840|
|  G|  GERARDO|          30|1168231104512|
|  I|     INMA|          25|1391569403904|
|  H| HERNANDO|          25|1683627180032|
+---+---------+------------+-------------+
```

- **Triangle count:** Cuenta el número de triángulos para cada nodo en el gráfico y calcula el coeficiente de agrupamiento promedio para la red de nodos resultante. Un triángulo se define como tres nodos que están conectados por tres bordes (a-b, b-c, c-a).

In [34]:
```
result = g.triangleCount()
result.show()
```

```
+-----+---+---------+------------+
|count| id|     name|total_points|
+-----+---+---------+------------+
|    0|  A|      ANA|         350|
|    0|  B|    BERTO|         360|
|    0|  C|    CLARA|         195|
|    0|  D|   DANIEL|          90|
|    0|  E|    ERICA|          90|
|    0|  F|FRANCISCO|         215|
|    0|  G|  GERARDO|          30|
|    0|  H| HERNANDO|          25|
|    0|  I|     INMA|          25|
|    0|  J|     JUAN|          20|
+-----+---+---------+------------+
```

No existe agrupamiento para ese ejemplo

- **Shortest paths:** Calcula la ruta más corta (ponderada) entre un par de nodos.

In [35]:
```
result = g.shortestPaths(landmarks=["B"])
result.show()
```

```
+---+---------+------------+---------+
| id|     name|total_points|distances|
+---+---------+------------+---------+
|  F|FRANCISCO|         215| {B -> 1}|
|  E|    ERICA|          90|       {}|
|  B|    BERTO|         360| {B -> 0}|
|  D|   DANIEL|          90| {B -> 2}|
|  C|    CLARA|         195|       {}|
|  J|     JUAN|          20|       {}|
|  A|      ANA|         350| {B -> 1}|
|  G|  GERARDO|          30|       {}|
|  I|     INMA|          25|       {}|
|  H| HERNANDO|          25|       {}|
+---+---------+------------+---------+
```

- **Breadth-first search (BFS):** Recorrer el gráfico desde el nodo raíz y explora todos los nodos vecinos. Luego, selecciona el nodo más cercano y explora todos los nodos inexplorados. El algoritmo sigue el mismo proceso para cada uno de los nodos más cercanos hasta que encuentra el objetivo.

In [36]:
```
paths = g.bfs("name='BERTO'", "total_points < 250")
paths.show()
```

```
+---------------+----------+------------------+
|           from|        e0|                to|
+---------------+----------+------------------+
|{B, BERTO, 360}|{B, E, 90}|     {E, ERICA, 90}|
|{B, BERTO, 360}|{B, F, 50}|{F, FRANCISCO, 215}|
+---------------+----------+------------------+
```

- **Subgraphs:** Selecciona los nodos que cumplen una serie de normas.

In [38]:
```
g2 = g.filterEdges("relationship = 50").filterVertices("total_points > 30
g2.vertices.show()
```

```
+---+---------+------------+
| id|     name|total_points|
+---+---------+------------+
|  A|      ANA|         350|
|  B|    BERTO|         360|
|  C|    CLARA|         195|
|  F|FRANCISCO|         215|
+---+---------+------------+
```

In [39]:
```
g2.edges.show()
```

```
+---+---+------------+
|src|dst|relationship|
+---+---+------------+
|  A|  C|          50|
|  B|  F|          50|
+---+---+------------+
```

- **Motif finding:** Se conoce como coincidencia de patrones de gráficos. La coincidencia de patrones encuentra algún patrón dentro del gráfico. El patrón es una expresión que se usa para definir algunos vértices conectados.

In [42]:
```
motifs = g.find("(a)-[e]->(b); (b)-[e2]->(a)")
motifs.show()
```

```
+------------------+----------+------------------+----------+
|                 a|         e|                 b|        e2|
+------------------+----------+------------------+----------+
|    {B, BERTO, 360}| {B, A, 60}|      {A, ANA, 350}| {A, B, 60}|
|    {D, DANIEL, 90}|{D, A, 100}|      {A, ANA, 350}| {A, D, 80}|
|      {A, ANA, 350}| {A, B, 60}|    {B, BERTO, 360}| {B, A, 60}|
|{F, FRANCISCO, 215}| {F, B, 60}|    {B, BERTO, 360}| {B, F, 50}|
|      {A, ANA, 350}| {A, D, 80}|    {D, DANIEL, 90}|{D, A, 100}|
|    {B, BERTO, 360}| {B, F, 50}|{F, FRANCISCO, 215}| {F, B, 60}|
+------------------+----------+------------------+----------+
```

Creado por:

Isabel Maniega