Assignment 3 Part 2: Analyzing Social Networks using GraphX/GraphFrame

Installing Spark and dependencies

Install Dependencies:

1. Java 8
2. Apache Spark with hadoop and
3. Findspark (used to locate the spark in the system)

```
!rm -rf spark-3.1.1-bin-hadoop3.2
!apt-get install openjdk-8-jdk-headless -qq > /dev/null
!pip install -q findspark pyspark
```

Set Environment Variables:

```
import os
os.environ["JAVA_HOME"] = "/usr/lib/jvm/java-8-openjdk-amd64"
```

Installing GraphFrames

```
!pip install graphframes
```

```
Requirement already satisfied: graphframes in /usr/local/lib/python3.10/dist-p
Requirement already satisfied: numpy in /usr/local/lib/python3.10/dist-package
Requirement already satisfied: nose in /usr/local/lib/python3.10/dist-packages
```

```
!curl -L -o "/usr/local/lib/python3.10/dist-packages/pyspark/jars/graphframes-0.8.2
```

```
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Curren
                                 Dload  Upload   Total   Spent    Left  Speed
100  242k  100  242k    0     0  1204k      0 --:--:-- --:--:-- --:--:-- 1210l
```

Downloading Wikipedia vote network

Dataset link: https://snap.stanford.edu/data/wiki-Vote.html

```
!wget https://snap.stanford.edu/data/wiki-Vote.txt.gz
!rm -f wiki-Vote.txt
!gzip -d wiki-Vote.txt.gz
```

```
--2024-07-20 15:20:19--  https://snap.stanford.edu/data/wiki-Vote.txt.gz
Resolving snap.stanford.edu (snap.stanford.edu)... 171.64.75.80
Connecting to snap.stanford.edu (snap.stanford.edu)|171.64.75.80|:443... conne
HTTP request sent, awaiting response... 200 OK
Length: 290339 (284K) [application/x-gzip]
Saving to: ‘wiki-Vote.txt.gz’

wiki-Vote.txt.gz    100%[===================>] 283.53K  --.-KB/s    in 0.1s

2024-07-20 15:20:20 (1.87 MB/s) - ‘wiki-Vote.txt.gz’ saved [290339/290339]
```

Importing libraries

```
import findspark
from pyspark.sql import SparkSession
from graphframes import *
from graphframes import GraphFrame
from pyspark.sql.functions import desc
```

Starting Spark and loading the dataset

```
findspark.init()

spark = (
    SparkSession.builder
    .config("spark.jars", "/usr/local/lib/python3.10/dist-packages/pyspark/jars/g
    .getOrCreate()
    )

spark.conf.set("spark.sql.repl.eagerEval.enabled", True)  # Property used to form
spark.sparkContext.setCheckpointDir("/tmp")
```

```
data = (
    spark
    .sparkContext
    .textFile("wiki-Vote.txt")
    .filter(lambda x: x[0] != "#")
    .map(lambda x: (x.split("\t")[0], x.split("\t")[1]))
    )
```

```
data.take(5)
```

```
[('30', '1412'),
 ('30', '3352'),
 ('30', '5254'),
 ('30', '5543'),
 ('30', '7478')]
```

```
vertices =  spark.createDataFrame(data
            .flatMap(lambda x: x)
            .distinct()
            .map(lambda x: (x,))
            .collect(),
             ["id"]
            )
vertices.show(5)
```

```
+----+
|  id|
+----+
|1412|
|3352|
|5254|
|5543|
|7478|
+----+
only showing top 5 rows
```

```
edges = spark.createDataFrame(data
        .flatMap(lambda x: [(x[0], x[1], "votes")])
        .distinct()
        .collect(),
        ["src", "dst", "type"]
        )
edges.show(5)
```

```
+---+---+-----+
|src|dst| type|
+---+---+-----+
|  3| 28|votes|
|  3| 30|votes|
|  3| 39|votes|
|  3|152|votes|
|  3|178|votes|
+---+---+-----+
only showing top 5 rows
```

Creating GraphFrame

```
graph = GraphFrame(vertices, edges).cache()
```

```
/usr/local/lib/python3.10/dist-packages/pyspark/sql/dataframe.py:168: UserWarn
  warnings.warn(
```

a. Find the top 5 nodes with the highest outdegree and find the count of the number of outgoing edges in each

```
outDegree = (
    graph
    .outDegrees
    .orderBy(desc("outDegree"))
    )
outDegree.show(5)
```

```
/usr/local/lib/python3.10/dist-packages/pyspark/sql/dataframe.py:147: UserWarn
  warnings.warn("DataFrame constructor is internal. Do not directly use it.")
+----+---------+
|  id|outDegree|
+----+---------+
|2565|      893|
| 766|      773|
|  11|      743|
| 457|      732|
|2688|      618|
+----+---------+
only showing top 5 rows
```

b. Find the top 5 nodes with the highest indegree and find the count of the number of incoming edges in each

```
inDegree = (
    graph
    .inDegrees
    .orderBy(desc("inDegree"))
    )
inDegree.show(5)
```

```
+----+--------+
|  id|inDegree|
+----+--------+
|4037|     457|
|  15|     361|
|2398|     340|
|2625|     331|
|1297|     309|
+----+--------+
only showing top 5 rows
```

c. Calculate PageRank for each of the nodes and output the top 5 nodes with the highest PageRank values. You are free to define any suitable parameters.

```
pageRank = (
    graph
    .pageRank(resetProbability=0.15, maxIter=5)
    .vertices
    .orderBy(desc("pagerank"))
    .select("id", "pagerank")
    )
pageRank.show(5)
```

```
+----+------------------+
|  id|          pagerank|
+----+------------------+
|4037|32.594991861511566|
|6634|29.983324850197288|
|  15| 27.01099302107744|
|2625|25.133148331225353|
|2398|  20.3858798963518|
+----+------------------+
only showing top 5 rows
```

d. Run the connected components algorithm on it and find the top 5 components with the largest number of nodes.

```
connectedComponents = (
    graph
    .connectedComponents()
    .select("id", "component")
    .groupBy("component")
    .count()
    .sort(desc("count"))
    )
connectedComponents.show(5)
```

```
+------------+-----+
|   component|count|
+------------+-----+
|           0| 7066|
|532575944741|    3|
|592705486870|    3|
|936302870556|    3|
|103079215124|    2|
+------------+-----+
only showing top 5 rows
```

```
stronglyConnectedComponents = (
    graph
    .stronglyConnectedComponents(maxIter=5)
    .select("id", "component")
    .groupBy("component")
    .count()
    .sort(desc("count"))
    )
stronglyConnectedComponents.show(5)
```

```
+---------+-----+
|component|count|
+---------+-----+
|        1| 1300|
|       26|    1|
|       19|    1|
|        0|    1|
|       22|    1|
+---------+-----+
only showing top 5 rows
```

e. Run the triangle counts algorithm on each of the vertices and output the top 5 vertices with the largest triangle count. In case of ties, you can randomly select the top 5 vertices.

```
triangleCount = (
    graph
    .triangleCount()
    .select("id", "count")
    .orderBy(desc("count"))
    )
triangleCount.show(5)
```

```
+----+-----+
|  id|count|
+----+-----+
|2565|30940|
|1549|22003|
| 766|18204|
|1166|17361|
|2688|14220|
+----+-----+
only showing top 5 rows
```

Creating Output files

```
outDegree.coalesce(1).write.csv("outDegree", header=True, mode="overwrite")
inDegree.coalesce(1).write.csv("inDegree", header=True, mode="overwrite")
pageRank.coalesce(1).write.csv("pageRank", header=True, mode="overwrite")
connectedComponents.coalesce(1).write.csv("connectedComponents", header=True, mode=
stronglyConnectedComponents.coalesce(1).write.csv("stronglyConnectedComponents", he
triangleCount.coalesce(1).write.csv("triangleCount", header=True, mode="overwrite")
```