LAB-2

Contributors

Member 1

- Name : Kavin Kumar Arumugam
- Class Id : 1

Member 2

- Name : Alper Erel
- Class Id : 8

Member 3

- Name : Jayden Tran
- Class Id : 27

Table of Contents:

1. Objectives

Question-1: Discovering Facebook basic companions:

Facebook has a rundown of companions (note that companions are a bi-directional thing on Facebook. In case I'm your companion, you're mine). They additionally have heaps of circle space and they serve a huge number of solicitations consistently. They have decided to pre-figure computations when they can to lessen the handling time of solicitations. One basic handling demand is the "You and Joe share 230 companions for all intents and purpose" include. At the point when you visit somebody's profile, you see a rundown of companions that you share for all intents and purpose. We're going to utilize MapReduce with the goal that we can compute everybody's regular companions once per day and store those outcomes. Later on,it's only a fast query. We have heaps of circle, it's modest.

Question-2: Spark Data Frames:

a. Make a Spark DataFrame utilizing one of datasets and attempt to utilize all unique StructType.
b. Perform 10 natural inquiries in Dataset (e.g.: design acknowledgment, subject dialog, most significant terms, and so forth.). Utilize your development to thoroughly consider of box.
c. Play out any 5 inquiries in Spark RDD's and Spark Data Frames. Think about the outcomes.(for example Choosing no of times Brazil won the World Cup , Selecting the Argentina WorldCup measurements and so forth)

Question-3: Spark Streaming TaskPerform:

Word-Count on Twitter Streaming Data utilizing Spark.

Question-4: Spark Graphx Task:

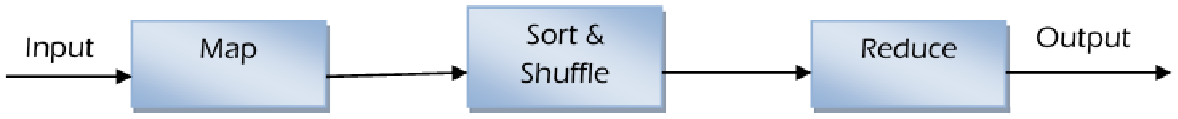
a.Perform Page Rank
b.State significance of utilizing graphx on the picked dataset.

2. Approaches/Methods

Question-1: Discovering Facebook basic companions:

Map Reduce calculation is beneficial with regards to process broad quantities of lines of information. It parts input task into tinier and sensible sub-assignments to oversee them in parallel. Guide Reduce calculation essentially sends the handling hub to where the information stands.

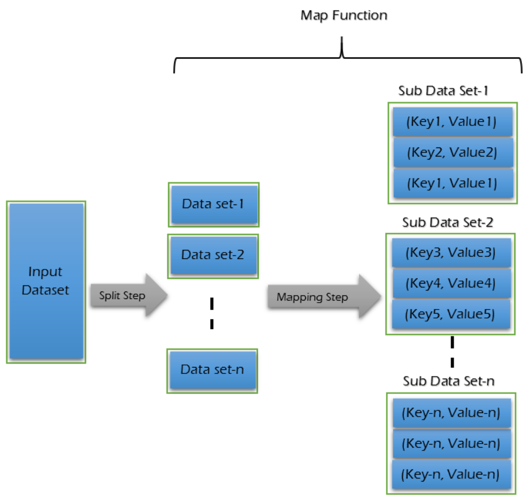
Three Stages of Map Reduce:



1) Map Stage

Guide Function is the underlying stage in MapReduce Algorithm. At Map stage will tackle key and worth sets as info.
A rundown of information is given to mapper class called mapper

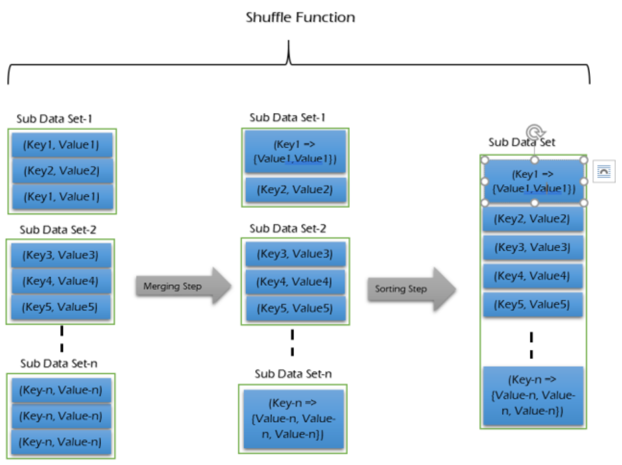
Splitting - Takes input dataset and separate the info dataset into little groups.
Mapping - Takes the splitted dataset and perform required calculation or activity on every one of them.
OUTPUT: set of key and worth combines as <Key, Value>.



2) Sort and Shuffle Stage

Rearrange and Sort is the second stage in MapReduce Algorithm.
This Shuffle and Sort is additionally called as "Join Function"
The yield from the mapper class is taken as information and sort and mix them.

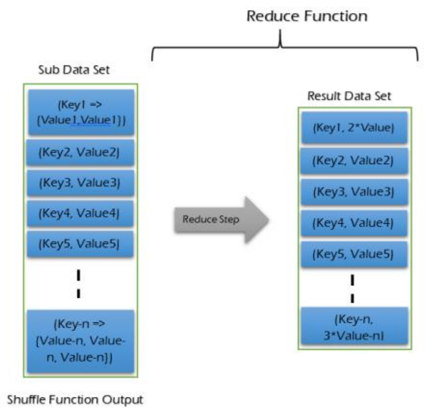
Merging - Find and union all key and worth sets which all have same key.
Sorting - Sort the entirety of the key and worth combines by keys.
OUTPUT: gathering of key and worth matches as <Key, List>



3) Reduce Stage

Reducer is the last stage in MapReduce Algorithm.
Takes rundown of arranged sets of <Key, List>
Subsequent to completing the reducer part the bunch gathers the information and send the information back to hadoop server.

OUTPUT: Result as <Key, Value>



Question-2: Spark Data Frames:

DataFrame:

- A Data Frame is utilized for putting away information in tables.
- It is proportionate to a table in a social database yet with more extravagant streamlining.
- It is an information deliberation and space explicit language (DSL) appropriate to a structure and semi-organized information.
  * It is an appropriated assortment of information as named section and column.
- It has a grid like structure whose section might be various sorts (numeric, legitimate, factor, or character).
- we can say information outline has a two-dimensional exhibit like structure where every segment contains the estimation of one variable and line contains one lot of qualities for every segment.
- It joins highlight of rundown and lattices.

RDD:

- RDD is the portrayal of a lot of records, permanent assortment of items with conveyed registering.
- RDD is a huge assortment of information or RDD is a variety of reference for apportioned articles.
- Every single dataset in RDD is consistently divided crosswise over numerous servers with the goal that they can be processed on various hubs of the group.
- RDDs are flaw tolerant for example self-recouped/recomputed on account of disappointment.
- The dataset could be information stacked remotely by the clients which can be as JSON document, CSV record, content document or database through JDBC with no particular information structure.

Question-3: Spark Streaming TaskPerform:

Twitter Streaming to TCP

- These days, information is developing and amassing quicker than at any other time. As of now, around 90% of all information produced in our reality was created distinctly over the most recent two years. Because of this stunning development rate, enormous information stages needed to embrace radical arrangements so as to keep up such immense volumes of information.
- One of the fundamental wellsprings of information today are interpersonal organizations. Enable me to exhibit a genuine model: Dealing, examining and removing bits of knowledge from informal organization information continuously utilizing one of the most significant large information reverberation arrangements out there—Apache Spark, and Python.
- **Step-1:** In this progression, the best way to manufacture a straightforward customer that will get the tweets from Twitter API utilizing Python and passes them to the Spark Streaming occasion. Import the libraries that we'll use as underneath:
- **Step-2:** Furthermore, include the factors that will be utilized in OAuth for interfacing with Twitter as underneath.
- **Step-3:** Presently, we will make another capacity called get_tweets that will call the Twitter API URL and return the reaction for a surge of tweets.
- **Step-4:** At that point, make a capacity that takes the reaction from the over one and concentrates the tweets' content from the entire tweets' JSON object. From that point forward, it sends each tweet to Spark Streaming occurrence (will be talked about later) through a TCP association.
- **Step-5:** Presently, we'll make the primary part which will make the application have attachment associations that flash will interface with. We'll design the IP here to be localhost as all will run on a similar machine and the port 9009. At that point we'll call the get_tweets technique, which we made above, for getting the tweets from Twitter and pass its reaction alongside the attachment association with send_tweets_to_spark for sending the tweets to Spark.

TCP to Spark Streaming

- **Step-1:** Setting Up Our Apache Spark Streaming Application. We should develop our Spark spilling application that will do ongoing handling for the approaching tweets, separate the hashtags from them, and compute what number of hashtags have been referenced. Initially, we need to make an occurrence of Spark Context sc, at that point we made the Streaming Context ssc from sc with a group interim two seconds that will do the change on all streams got like clockwork. Notice we have set the log level to ERROR so as to incapacitate a large portion of the logs that Spark composes. We characterized a checkpoint here so as to permit occasional RDD checkpointing; this is required to be utilized in our application, as we'll utilize stateful changes (will be talked about later in a similar area). At that point we characterize our fundamental DStream dataStream that will associate with the attachment server we made before on port 9009 and read the tweets from that port. Each record in the DStream will be a tweet.
- **Step-2:** Presently, we'll characterize our change rationale. First we'll part every one of the tweets into words and put them in words RDD. At that point we'll channel just hashtags from all words and guide them to combine of (hashtag, 1) and put them in hashtags RDD. At that point we have to figure how frequently the hashtag has been referenced. We can do that by utilizing the capacity reduceByKey.
- **Step-3:** The updateStateByKey accepts a capacity as a parameter called the update work. It runs on every thing in RDD and does the ideal rationale. For our situation, we've made an update work called aggregate_tags_count that will entirety all the new_values for each hashtag and add them to the total_sum that is the total over every one of the clumps and spare the information into tags_totals RDD. At that point we do handling on tags_totals RDD in each cluster so as to change over it to temp table utilizing Spark SQL Context and afterward play out a select articulation so as to recover the best ten hashtags with their tallies and put them into hashtag_counts_df information outline.

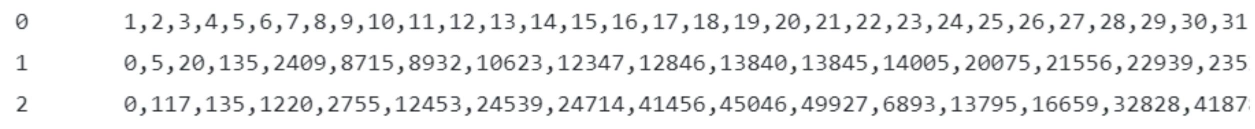
Question-4: Spark Graphx Task:

GraphX broadens the Spark RDD with a Resilient Distributed Property Graph. The property diagram is a coordinated multigraph which can have numerous edges in parallel. Each edge and vertex has client characterized properties related with it. The parallel edges permit different connections between a similar vertices.

3. Datasets

Question-1: Discovering Facebook basic companions:

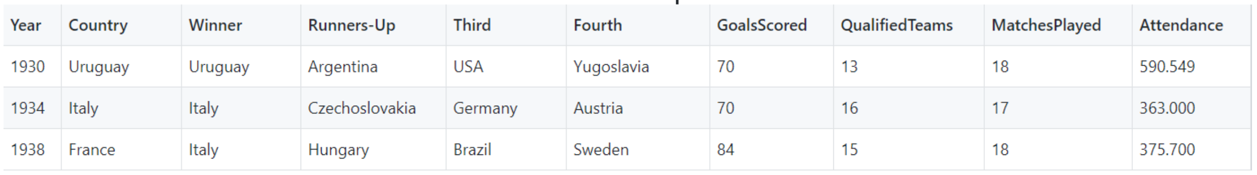
- To see the dataset -> click here to see Facebook Friends Dataset.



```
0       1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27,28,29,30,31
1       0,5,20,135,2409,8715,8932,10623,12347,12846,13840,13845,14005,20075,21556,22939,235
2       0,117,135,1220,2755,12453,24539,24714,41456,45046,49927,6893,13795,16659,32828,4187
```

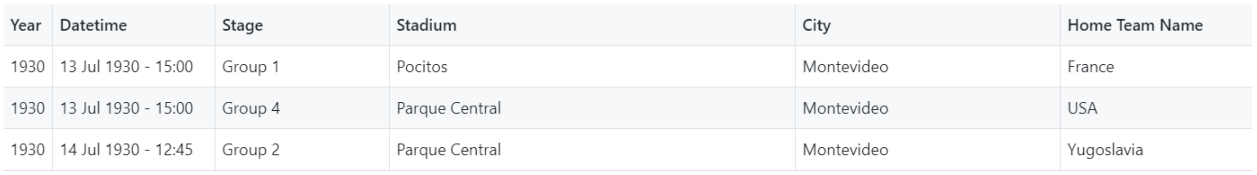
Question-2: Spark Data Frames:

- To see the dataset -> click here to see WorldCups Dataset.



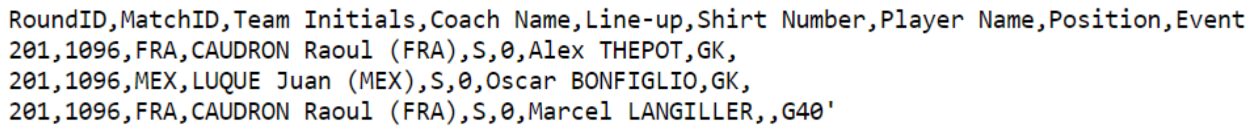
Year	Country	Winner	Runners-Up	Third	Fourth	GoalsScored	QualifiedTeams	MatchesPlayed	Attendance
1930	Uruguay	Uruguay	Argentina	USA	Yugoslavia	70	13	18	590.549
1934	Italy	Italy	Czechoslovakia	Germany	Austria	70	16	17	363.000
1938	France	Italy	Hungary	Brazil	Sweden	84	15	18	375.700

- To see the dataset -> click here to see WorldCupsMatches Dataset.



Year	Datetime	Stage	Stadium	City	Home Team Name
1930	13 Jul 1930 - 15:00	Group 1	Pocitos	Montevideo	France
1930	13 Jul 1930 - 15:00	Group 4	Parque Central	Montevideo	USA
1930	14 Jul 1930 - 12:45	Group 2	Parque Central	Montevideo	Yugoslavia

- To see the dataset -> click here to see WorldCupsPlayers Dataset.



```
RoundID,MatchID,Team Initials,Coach Name,Line-up,Shirt Number,Player Name,Position,Event
201,1096,FRA,CAUDRON Raoul (FRA),S,0,Alex THEPOT,GK,
201,1096,MEX,LUQUE Juan (MEX),S,0,Oscar BONFIGLIO,GK,
201,1096,FRA,CAUDRON Raoul (FRA),S,0,Marcel LANGILLER,,G40'
```

Question-3: Spark Streaming TaskPerform:

We haven't used any specific dataset to perform Spark Streaming we did this from the Streaming Context.

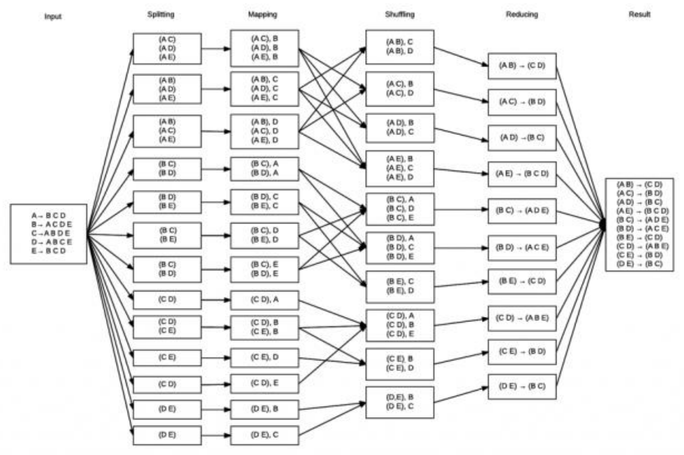
Question-4: Spark Graphx Task:

Used WordGame dataset.

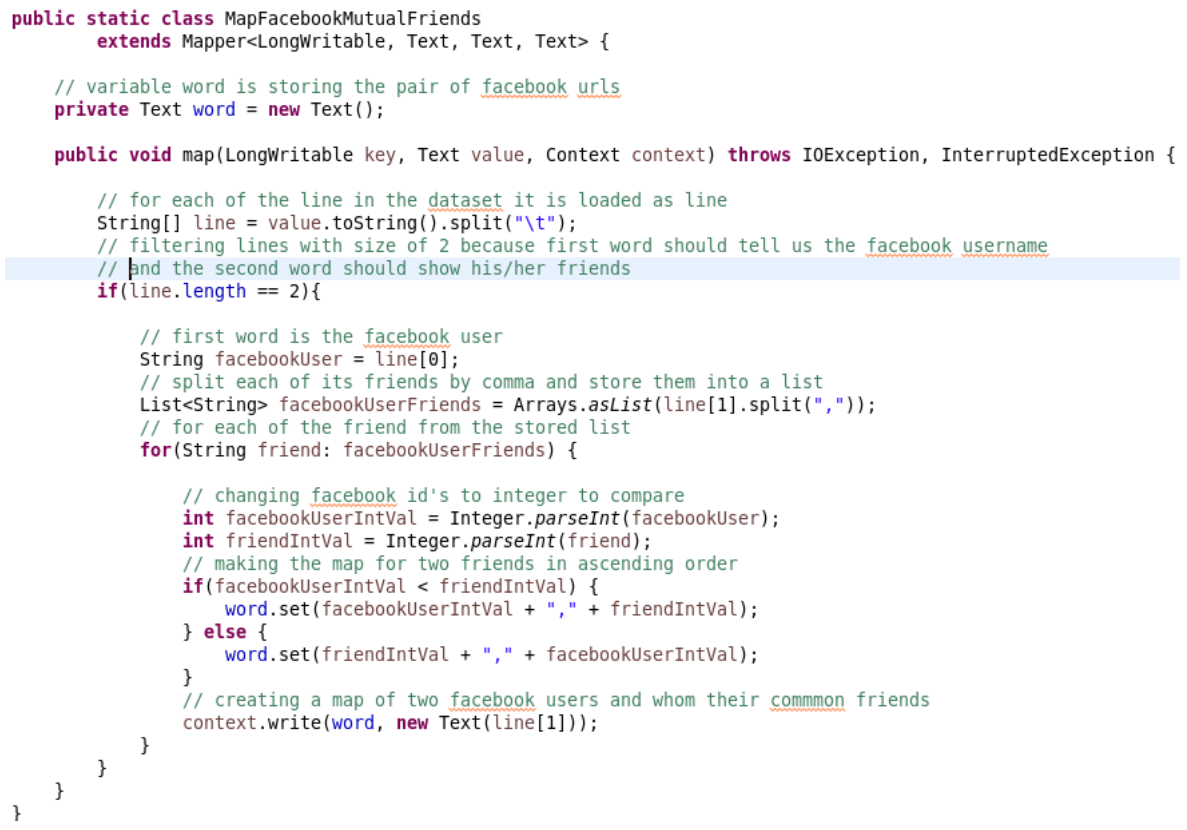
4. Code Screenshots

Question-1: Discovering Facebook basic companions:

Map Reduce Diagram:

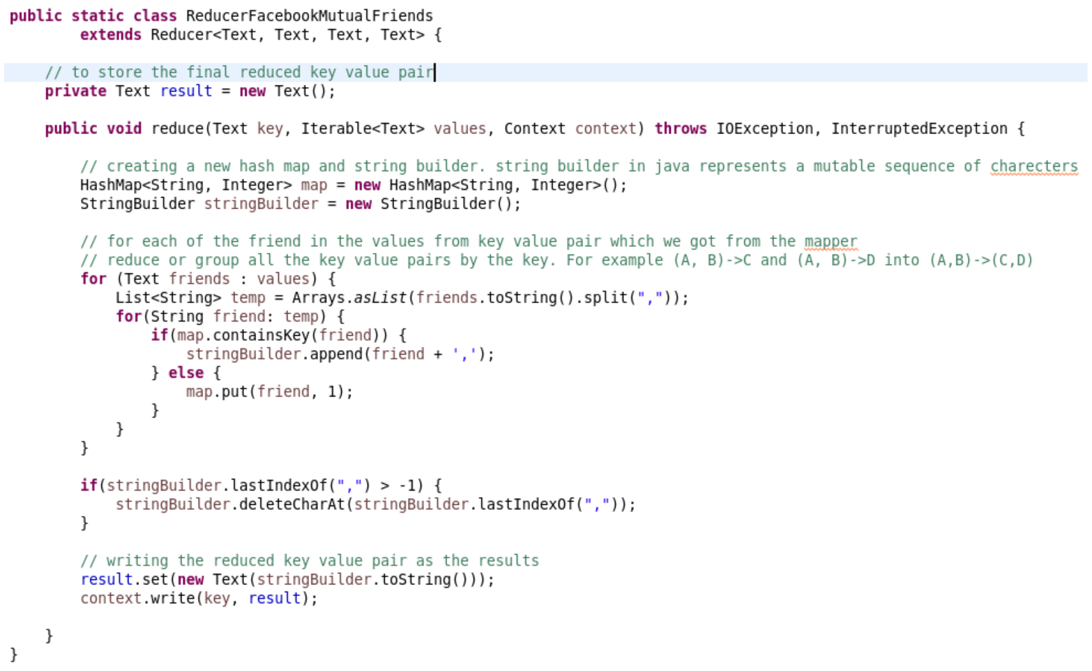


1. Mapper (To see the code -> click here)



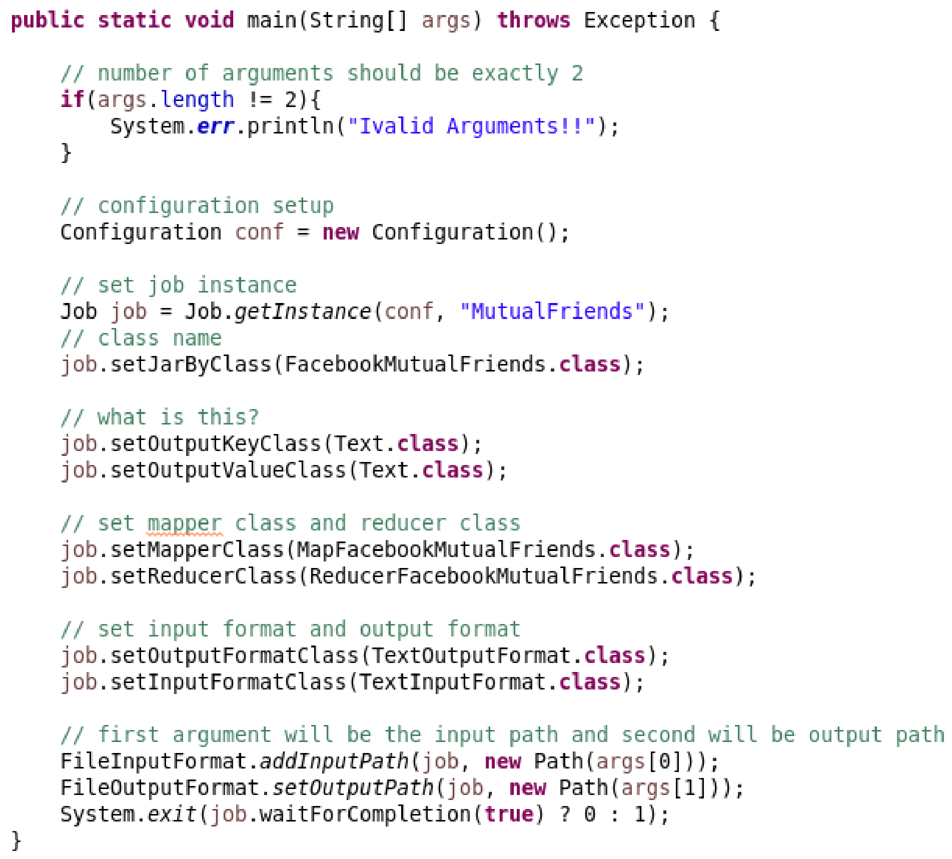
```
public static class MapFacebookMutualFriends
        extends Mapper<LongWritable, Text, Text, Text> {

    // variable word is storing the pair of facebook urls
    private Text word = new Text();

    public void map(LongWritable key, Text value, Context context) throws IOException, InterruptedException {

        // for each of the line in the dataset it is loaded as line
        String[] line = value.toString().split("\t");
        // filtering lines with size of 2 because first word should tell us the facebook username
        // and the second word should show his/her friends
        if(line.length == 2){

            // first word is the facebook user
            String facebookUser = line[0];
            // split each of its friends by comma and store them into a list
            List<String> facebookUserFriends = Arrays.asList(line[1].split(","));
            // for each of the friend from the stored list
            for(String friend: facebookUserFriends) {

                // changing facebook id's to integer to compare
                int facebookUserIntVal = Integer.parseInt(facebookUser);
                int friendIntVal = Integer.parseInt(friend);
                // making the map for two friends in ascending order
                if(facebookUserIntVal < friendIntVal) {
                    word.set(facebookUserIntVal + "," + friendIntVal);
                } else {
                    word.set(friendIntVal + "," + facebookUserIntVal);
                }
                // creating a map of two facebook users and whom their commmon friends
                context.write(word, new Text(line[1]));
            }
        }
    }
}
```

2. Reducer (To see the code -> click here)



```
public static class ReducerFacebookMutualFriends
        extends Reducer<Text, Text, Text, Text> {

    // to store the final reduced key value pair
    private Text result = new Text();

    public void reduce(Text key, Iterable<Text> values, Context context) throws IOException, InterruptedException {

        // creating a new hash map and string builder. string builder in java represents a mutable sequence of charecters
        HashMap<String, Integer> map = new HashMap<String, Integer>();
        StringBuilder stringBuilder = new StringBuilder();

        // for each of the friend in the values from key value pair which we got from the mapper
        // reduce or group all the key value pairs by the key. For example (A, B)->C and (A, B)->D into (A,B)->(C,D)
        for (Text friends : values) {
            List<String> temp = Arrays.asList(friends.toString().split(","));
            for(String friend: temp) {
                if(map.containsKey(friend)) {
                    stringBuilder.append(friend + ',');
                } else {
                    map.put(friend, 1);
                }
            }
        }

        if(stringBuilder.lastIndexOf(",") > -1) {
            stringBuilder.deleteCharAt(stringBuilder.lastIndexOf(","));
        }

        // writing the reduced key value pair as the results
        result.set(new Text(stringBuilder.toString()));
        context.write(key, result);

    }
}
```

3. Main Class (To see the code -> click here)



```
public static void main(String[] args) throws Exception {

    // number of arguments should be exactly 2
    if(args.length != 2){
        System.err.println("Ivalid Arguments!!");
    }

    // configuration setup
    Configuration conf = new Configuration();

    // set job instance
    Job job = Job.getInstance(conf, "MutualFriends");
    // class name
    job.setJarByClass(FacebookMutualFriends.class);

    // what is this?
    job.setOutputKeyClass(Text.class);
    job.setOutputValueClass(Text.class);

    // set mapper class and reducer class
    job.setMapperClass(MapFacebookMutualFriends.class);
    job.setReducerClass(ReducerFacebookMutualFriends.class);

    // set input format and output format
    job.setOutputFormatClass(TextOutputFormat.class);
    job.setInputFormatClass(TextInputFormat.class);

    // first argument will be the input path and second will be output path
    FileInputFormat.addInputPath(job, new Path(args[0]));
    FileOutputFormat.setOutputPath(job, new Path(args[1]));
    System.exit(job.waitForCompletion(true) ? 0 : 1);
}
```

Question-2: Spark Data Frames:

Step-1: Importing Libraries:

```
from pyspark.sql import SparkSession
from pyspark.sql import *
from pyspark.sql.types import DoubleType
```

Step-2: Creating Spark Session:

```
spark = SparkSession \
    .builder \
    .appName("Python Spark SQL basic example") \
    .config("spark.some.config.option", "some-value") \
    .getOrCreate()
```

Step-3: Creating Spark Dataframe:

```
df = spark.read.csv("WorldCupMatches.csv",header=True);
df.show()
print(df.schema)
```

Step-4: Verifying Spark Dataframe:

```
+----+--------------------+-----------------+--------------------+-----------+--------------+---------------+---------------+--
------------+--------------------+-----------+--------------------+--------------------+--------------------+-------------------
-+--------------------+-------+-------+------------------+-----------------+
|Year|            Datetime|            Stage|             Stadium|       City|Home Team Name|Home Team Goals|Away Team Goals|Aw
ay Team Name|      Win conditions|Attendance|Half-time Home Goals|Half-time Away Goals|             Referee|          Assistant
1|          Assistant 2|RoundID|MatchID|Home Team Initials|Away Team Initials|
+----+--------------------+-----------------+--------------------+-----------+--------------+---------------+---------------+--
------------+--------------------+-----------+--------------------+--------------------+--------------------+-------------------
-+--------------------+-------+-------+------------------+-----------------+
|1930|13 Jul 1930 - 15:00 |          Group 1|             Pocitos|Montevideo |        France|              4|              1|
Mexico|                    |      4444|                   3|                   0|LOMBARDI Domingo ...|CRISTOPHE Henry (...| REG
O Gilberto (BRA)|    201|   1096|               FRA|              MEX|
|1930|13 Jul 1930 - 15:00 |          Group 4|      Parque Central|Montevideo |           USA|              3|              0|
Belgium|                    |     18346|                   2|                   0|   MACIAS Jose (ARG)|MATEUCCI Francisc...|WAR
```

Query-1: Renaming Column Names.

```
df.columns
```

```
['Year',
 'Datetime',
 'Stage',
 'Stadium',
 'City',
 'Home Team Name',
 'Home Team Goals',
 'Away Team Goals',
 'Away Team Name',
 'Win conditions',
 'Attendance',
 'Half-time Home Goals',
 'Half-time Away Goals',
 'Referee',
 'Assistant 1',
 'Assistant 2',
 'RoundID',
 'MatchID',
 'Home Team Initials',
 'Away Team Initials']
```

```
df=df.withColumnRenamed('Home Team Name', 'Home_Team')
df=df.withColumnRenamed('Away Team Name', 'Away_Team')
df=df.withColumnRenamed('Away Team Goals', 'Away_Team_Goals')
df=df.withColumnRenamed('Home Team Goals', 'Home_Team_Goals')
df=df.withColumnRenamed('Assistant 1', 'Assistant_1')
df=df.withColumnRenamed('Assistant 2', 'Assistant_2')
df=df.withColumnRenamed('Home Team Initials', 'Home_Team_Initials')
df=df.withColumnRenamed('Away Team Initials', 'Away_Team_Initials')
df=df.withColumnRenamed('Half-time Home Goals', 'Half-time_Home_Goals')
df=df.withColumnRenamed('Half-time Away Goals', 'Half-time_Away_Goals')
```

```
df.columns
```

Query-2: Changing Datatypes of Dataframe.

```
['Year',
 'Datetime',
 'Stage',
 'Stadium',
 'City',
 'Home_Team',
 'Home_Team_Goals',
 'Away_Team_Goals',
 'Away_Team',
 'Win conditions',
 'Attendance',
 'Half-time_Home_Goals',
 'Half-time_Away_Goals',
 'Referee',
 'Assistant_1',
 'Assistant_2',
 'RoundID',
 'MatchID',
 'Home_Team_Initials',
 'Away_Team_Initials']
```

```
df=df.withColumn('Away_Team_Goals',df['Away_Team_Goals'].cast(DoubleType()))
df=df.withColumn('Home_Team_Goals',df['Home_Team_Goals'].cast(DoubleType()))
df=df.withColumn('Attendance',df['Attendance'].cast(DoubleType()))
df=df.withColumn('Home_Team_Goals',df['Home_Team_Goals'].cast(DoubleType()))
df=df.withColumn('Away_Team_Goals',df['Away_Team_Goals'].cast(DoubleType()))
```

Query-3: Select year from the dataframe.

```
df.describe(['Year']).show()
```

```
+-------+------------------+
|summary|              Year|
+-------+------------------+
|  count|               852|
|   mean|1985.0892018779343|
| stddev|22.448824702021138|
|    min|              1930|
|    max|              2014|
+-------+------------------+
```

Query-4: Print number of matches where Czechoslovakia and Italy played for finals and semi-finals.

```
df.filter((df.Stage.like("Final") | df.Stage.like("Semi-finals")) & (df.Away_Team == 'Czechoslovakia') & (df.Home_Team == 'Italy
```

```
Out[38]:  1
```

Query-5: Print top 10 matches played in the year of 2014 and 2013.

```
df.filter(df.Year.like("2014") | df.Year.like("2013")).show(10)
```

```
+----+--------------------+-------+------------------+---------------+-------------+---------------+---------------+-----------
+--------------+----------+--------------------+--------------------+--------------------+--------------------+---------------
----+-------+----------+------------------+------------------+
|Year|            Datetime|  Stage|           Stadium|           City|    Home_Team|Home_Team_Goals|Away_Team_Goals|   Away_Team
|Win conditions|Attendance|Half-time_Home_Goals|Half-time_Away_Goals|             Referee|         Assistant_1|        Assista
nt_2|RoundID|   MatchID|Home_Team_Initials|Away_Team_Initials|
+----+--------------------+-------+------------------+---------------+-------------+---------------+---------------+-----------
+--------------+----------+--------------------+--------------------+--------------------+--------------------+---------------
----+-------+----------+------------------+------------------+
|2014|12 Jun 2014 - 17:00 |Group A|Arena de Sao Paulo|     Sao Paulo |       Brazil|            3.0|            1.0|     Croatia
|              |   62103.0|                   1|                   1|NISHIMURA Yuichi ...|   SAGARA Toru (JPN)|NAGI Toshiyuki
(JPN)| 255931|300186456|               BRA|               CRO|
|2014|13 Jun 2014 - 13:00 |Group A| Estadio das Dunas|         Natal |       Mexico|            1.0|            0.0|    Cameroon
|              |   39216.0|                   0|                   0| ROLDAN Wilmar (COL)|CLAVIJO Humberto ...|  DIAZ Eduardo
```

Query-6: Number of matches played in the year of 2014 and 2013.

```
print("Matches in 2014 and 2013 : " + str(df.filter(df.Year.like("2014") | df.Year.like("2013")).count()))
```

```
Matches in 2014 and 2013 : 80
```

Query-7: Number of matches played in each city.

```
df.groupBy('City').count().show(15,truncate=True)
```

```
+-------------------+-----+
|               City|count|
+-------------------+-----+
|              Daegu |    4|
|              Paris |    9|
|              Natal |    4|
|      San Francisco |    6|
|Santiago De Chile |   10|
|         Eskilstuna |    1|
|          La Coru�A |    3|
|             Bilbao |    3|
|             Geneva |    4|
|           Le Havre |    1|
|             Verona |    4|
|               Kobe |    3|
|              Solna |    8|
|          Liverpool |    5|
|            Gwangju |    3|
+-------------------+-----+
only showing top 15 rows
```

Query-8: Maximum number of goals scored by the Home Team.

```
df.groupBy('Home_Team').max('Home_Team_Goals').dropna().show()
```

```
+---------+--------------------+
|Home_Team|max(Home_Team_Goals)|
+---------+--------------------+
| Paraguay|                 3.0|
|   Russia|                 6.0|
|  Senegal|                 3.0|
|   Sweden|                 8.0|
|  IR Iran|                 0.0|
|   Turkey|                 7.0|
|    Zaire|                 0.0|
|     Iraq|                 1.0|
|  Germany|                 8.0|
|   France|                 7.0|
|   Greece|                 2.0|
|  Algeria|                 3.0|
|     Togo|                 0.0|
| Slovakia|                 3.0|
|Argentina|                 6.0|
|    Wales|                 2.0|
|  Belgium|                 3.0|
|   Angola|                 0.0|
|  Ecuador|                 3.0|
|    Ghana|                 2.0|
+---------+--------------------+
only showing top 20 rows
```

Query-9: Order by sum of away team goals.

```
df.groupBy('Away_Team').sum('Away_Team_Goals').orderBy('Away_Team').dropna().show(15,truncate=True)
```

```
+--------------------+--------------------+
|           Away_Team|sum(Away_Team_Goals)|
+--------------------+--------------------+
|"rn"">Bosnia and ...|                 1.0|
|"rn"">Republic of...|                 8.0|
|"rn"">Serbia and ...|                 2.0|
|"rn"">Trinidad an...|                 0.0|
|"rn"">United Arab...|                 2.0|
|             Algeria|                 9.0|
|              Angola|                 1.0|
|           Argentina|                22.0|
|           Australia|                 4.0|
|             Austria|                12.0|
|             Belgium|                27.0|
|             Bolivia|                 0.0|
|              Brazil|                45.0|
|            Bulgaria|                11.0|
|            Cameroon|                 7.0|
+--------------------+--------------------+
only showing top 15 rows
```

Query-10: CrossTab on the columns Year, Home_Team and Stage.

```
from pyspark.sql.functions import col, struct

df.withColumn("Home_Team_By_Year", struct("Year", "Home_Team")).crosstab("Home_Team_By_Year", "Stage")
```

```
DataFrame[Home_Team_By_Year_Stage: string, Final: bigint, First round: bigint, Group 1: bigint, Group 2: bigint, Group 3: bigin
t, Group 4: bigint, Group 5: bigint, Group 6: bigint, Group A: bigint, Group B: bigint, Group C: bigint, Group D: bigint, Group
E: bigint, Group F: bigint, Group G: bigint, Group H: bigint, Match for third place: bigint, Play-off for third place: bigint,
Preliminary round: bigint, Quarter-finals: bigint, Round of 16: bigint, Semi-finals: bigint, Third place: bigint, null: bigint]
```

Creating SparkContext for WorldCups

```
from pyspark import SparkContext

sc = SparkContext.getOrCreate()

lines = sc.textFile("WorldCups.csv", 1)
header = lines.first()
content = lines.filter(lambda line: line != header)

rdd = content.map(lambda line: (line.split(","))).collect()

rdd_len = content.map(lambda line: len(line.split(","))).distinct().collect()

print(rdd_len)
```

```
[10]
```

Changing the Schema

```
from pyspark.sql.types import StructType
from pyspark.sql.types import StructField
from pyspark.sql.types import StringType,IntegerType

schema = StructType([StructField('Year', StringType(), True),
StructField('Country', StringType(),True),
StructField('Winner', StringType(), True),
StructField('Runners-Up', StringType(),True),
StructField('Third', StringType(),True),
StructField('Fourth', StringType(),True),
StructField('GoalsScored', StringType(),True),
StructField('QualifiedTeams', StringType(),True),
StructField('MatchesPlayed', StringType(),True),
StructField('Attendance', StringType(),True)])
```

Query-1: Order by goals scored by each country.

```
print("---------- 1. Venues & goals scored ------------")

# venue - hosted country with highest goals (From RDD)
rdd1 = (content.filter(lambda line: line.split(",")[6] != "NULL")
.map(lambda line: (line.split(",")[1], int(line.split(",")[6])))
.takeOrdered(10, lambda x : -x[1]))
print(rdd1)


# Create data frame from the RDD
df = spark.createDataFrame(rdd,schema)

df=df.withColumn('GoalsScored',df['GoalsScored'].cast(IntegerType()))

# venue - hosted country with highest goals (From DF)
df.select("Country","GoalsScored").orderBy("GoalsScored", ascending = False).show(20, truncate = False)

# venue - hosted country with highest goals (From DF - SQL)
df.createOrReplaceTempView("df_table")
spark.sql(" SELECT Country,GoalsScored FROM df_table order by "+
"GoalsScored Desc Limit 10").show()
```

```
---------- 1. Venues & goals scored ------------
[('France', 171), ('Brazil', 171), ('Korea/Japan', 161), ('Germany', 147), ('Spain', 146), ('South Africa', 145), ('USA', 141),
('Switzerland', 140), ('Mexico', 132), ('Sweden', 126)]
+------------+-----------+
|Country     |GoalsScored|
+------------+-----------+
|France      |171        |
|Brazil      |171        |
|Korea/Japan |161        |
|Germany     |147        |
|Spain       |146        |
|South Africa|145        |
|USA         |141        |
|Switzerland |140        |
|Mexico      |132        |
|Sweden      |126        |
|Italy       |115        |
|Argentina   |102        |
|Germany     |97         |
|Mexico      |95         |
|England     |89         |
|Chile       |89         |
|Brazil      |88         |
|France      |84         |
|Uruguay     |70         |
|Italy       |70         |
+------------+-----------+



+------------+-----------+
|     Country|GoalsScored|
+------------+-----------+
|      France|        171|
|      Brazil|        171|
| Korea/Japan|        161|
|     Germany|        147|
|       Spain|        146|
|South Africa|        145|
|         USA|        141|
| Switzerland|        140|
|      Mexico|        132|
|      Sweden|        126|
+------------+-----------+
```

Query-2: Details of years starting from 2010.

```
print("-------------- 2. Details of years starting in 2010 ---------------")

# using RDD
years = ["2010", "2011", "2012", "2013", "2014"]
(content.filter(lambda line: line.split(",")[0] in years)
.map(lambda line: (line.split(",")[0],line.split(",")[2],line.split(",")[4])).collect())

# using DF
df.select("Year","Winner","Third").filter(df.Year.isin(years)).show()

# using DF - SQL
spark.sql(" SELECT Year,Winner,Third FROM df_table  WHERE " + " Year IN ('2010','2011','2012','2013','2014') ").show()
```

```
-------------- 3. Details of years starting in 2010 ---------------
+----+-------+-----------+
|Year| Winner|      Third|
+----+-------+-----------+
|2010|  Spain|    Germany|
|2014|Germany|Netherlands|
+----+-------+-----------+

+----+-------+-----------+
|Year| Winner|      Third|
+----+-------+-----------+
|2010|  Spain|    Germany|
|2014|Germany|Netherlands|
+----+-------+-----------+
```

Query-3: Select year, country and place where the place is third and order by year.

```
print("---------- 3. Year, venue country = Third Place ------------")

# using RDD
(content.filter(lambda line:line.split(",")[1]==line.split(",")[4])
 .map(lambda line: (line.split(",")[0],line.split(",")[1], line.split(",")[4]))
 .collect())

# using DF
df.select("Year","Country","Third").filter(df["Country"]==df["Third"]).show()

# using DF - SQL
spark.sql(" SELECT Year,Country,Third FROM df_table where Country == Third order by Year").show()
```

```
---------- 2. Year, venue country = Third Place ------------
+----+-------+-------+
|Year|Country|  Third|
+----+-------+-------+
|1962|  Chile|  Chile|
|1990|  Italy|  Italy|
|2006|Germany|Germany|
+----+-------+-------+

+----+-------+-------+
|Year|Country|  Third|
+----+-------+-------+
|1962|  Chile|  Chile|
|1990|  Italy|  Italy|
|2006|Germany|Germany|
+----+-------+-------+
```

Query-4: Max number of goals scored.

```
print("------------- 4. Max Goals Scored -----------------")

# Using RDD
(content.filter(lambda line:line.split(",")[6] == "171")
.map(lambda line: (line.split(","))).collect())

# using DF
df=df.withColumn('GoalsScored',df['GoalsScored'].cast(IntegerType()))
df.filter(df.GoalsScored == 171).show()

# using DF - SQL
spark.sql(" Select * from df_table where GoalsScored in " +
"(Select Max(GoalsScored) from df_table )" ).show()
```

```
------------- 4. Max matches played -----------------
+----+-------+-------+----------+----------+-----------+-----------+--------------+-------------+----------+
|Year|Country| Winner|Runners-Up|     Third|     Fourth|GoalsScored|QualifiedTeams|MatchesPlayed|Attendance|
+----+-------+-------+----------+----------+-----------+-----------+--------------+-------------+----------+
|1998| France| France|    Brazil|   Croatia|Netherlands|        171|            32|           64| 2.785.100|
|2014| Brazil|Germany| Argentina|Netherlands|     Brazil|        171|            32|           64| 3.386.810|
+----+-------+-------+----------+----------+-----------+-----------+--------------+-------------+----------+

+----+-------+-------+----------+----------+-----------+-----------+--------------+-------------+----------+
|Year|Country| Winner|Runners-Up|     Third|     Fourth|GoalsScored|QualifiedTeams|MatchesPlayed|Attendance|
+----+-------+-------+----------+----------+-----------+-----------+--------------+-------------+----------+
|1998| France| France|    Brazil|   Croatia|Netherlands|        171|            32|           64| 2.785.100|
|2014| Brazil|Germany| Argentina|Netherlands|     Brazil|        171|            32|           64| 3.386.810|
+----+-------+-------+----------+----------+-----------+-----------+--------------+-------------+----------+
```

Query-5: 2010 World cup statistics.

```
print("-------------- 5. 2010 world cup stats --------------")
# using RDD
(content.filter(lambda line:line.split(",")[0]=="2010")
.map(lambda line: (line.split(","))).collect())
# using DF
df.filter(df.Year=="2010").show()
# using DF - Sql
spark.sql(" Select * from df_table where Year == 2010 ").show()
```

```
------------- 4. 2010 world cup stats -------------
+----+------------+------+-----------+-------+-------+-----------+--------------+-------------+----------+
|Year|     Country|Winner| Runners-Up|  Third| Fourth|GoalsScored|QualifiedTeams|MatchesPlayed|Attendance|
+----+------------+------+-----------+-------+-------+-----------+--------------+-------------+----------+
|2010|South Africa| Spain|Netherlands|Germany|Uruguay|        145|            32|           64| 3.178.856|
+----+------------+------+-----------+-------+-------+-----------+--------------+-------------+----------+

+----+------------+------+-----------+-------+-------+-----------+--------------+-------------+----------+
|Year|     Country|Winner| Runners-Up|  Third| Fourth|GoalsScored|QualifiedTeams|MatchesPlayed|Attendance|
+----+------------+------+-----------+-------+-------+-----------+--------------+-------------+----------+
|2010|South Africa| Spain|Netherlands|Germany|Uruguay|        145|            32|           64| 3.178.856|
+----+------------+------+-----------+-------+-------+-----------+--------------+-------------+----------+
```

Question-3: Spark Streaming TaskPerform:

Twitter Streaming to TCP:

Getting tweets from twitter streaming.

```
def get_tweets():
    url = 'https://stream.twitter.com/1.1/statuses/filter.json'
    query_data = [('language', 'en'),('track','#')]
    query_url = url + '?' + '&'.join([str(t[0]) + '=' + str(t[1]) for t in query_data])
    response = requests.get(query_url, auth=my_auth, stream=True)
    print(query_url, response)
    return response
```

Sending tweets to Spark:

```
def send_tweets_to_spark(http_resp, tcp_connection):
    for line in http_resp.iter_lines():
        try:
            full_tweet = json.loads(line)
            tweet_text = full_tweet['text']
            print("Tweet Text: " + tweet_text)
            print ("------------------------------------------")
            tweet_data = bytes(tweet_text + "/n", "utf-8")
            tcp_connection.send(tweet_data)
        except:
            e = sys.exc_info()[0]
            print("Error: %s" % e)
```

Streaming to TCP:

```
TCP_IP = "localhost"
TCP_PORT = 9020
conn = None
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.bind((TCP_IP, TCP_PORT))
s.listen(1)
print("Waiting for TCP connection...")
conn, addr = s.accept()
print("Connected... Starting getting tweets.")
resp = get_tweets()
send_tweets_to_spark(resp, conn)
```

TCP to Spark Streaming:

Creating SparkContext and SparkStreamingContext:

```python
from pyspark import SparkConf,SparkContext
from pyspark.streaming import StreamingContext
from pyspark.sql import Row,SQLContext
import sys
import requests


# create spark configuration
conf = SparkConf()
conf.setAppName("TwitterStreamApp")
# create spark context with the above configuration
sc = SparkContext(conf=conf)
sc.setLogLevel("ERROR")
# create the Streaming Context from the above spark context with interval size 2 seconds
ssc = StreamingContext(sc, 2)
# setting a checkpoint to allow RDD recovery
ssc.checkpoint("checkpoint_TwitterApp")
# read data from port 9009
dataStream = ssc.socketTextStream("localhost",9020)
```

Aggregating the tags count:

```python
def aggregate_tags_count(new_values, total_sum):
    return sum(new_values) + (total_sum or 0)
```

Getting SQL context instance:

```python
def get_sql_context_instance(spark_context):
    if ('sqlContextSingletonInstance' not in globals()):
        globals()['sqlContextSingletonInstance'] = SQLContext(spark_context)
    return globals()['sqlContextSingletonInstance']
```

Processing each RDD:

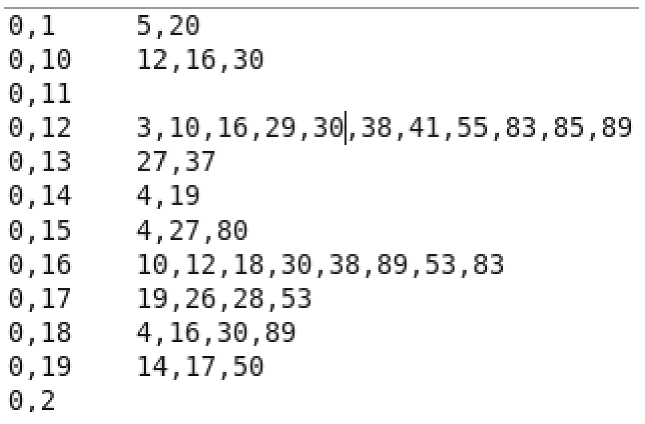
```python
def process_rdd(time, rdd):
    print("----------- %s -----------" % str(time))
    try:
        # Get spark sql singleton context from the current context
        sql_context = get_sql_context_instance(rdd.context)

        # convert the RDD to Row RDD
        row_rdd = rdd.map(lambda w: Row(hashtag=w[0], hashtag_count=w[1]))

        # create a DF from the Row RDD
        hashtags_df = sql_context.createDataFrame(row_rdd)
        # Register the dataframe as table
        hashtags_df.registerTempTable("hashtags")
        # get the top 10 hashtags from the table using SQL and print them
        hashtag_counts_df = sql_context.sql("select hashtag, hashtag_count from hashtags order by hashtag_count desc limit 10")
        hashtag_counts_df.show()
```

Question-4: Spark Graphx Task:

Creating spark dataframe for world game.

5. Results/Output Screenshots

Question-1: Discovering Facebook basic companions:

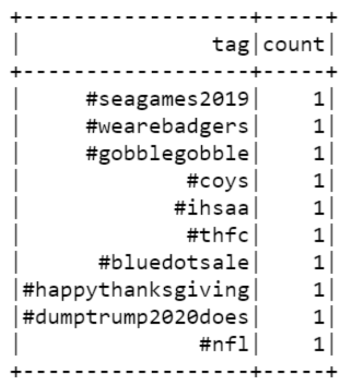


```
0,1     5,20
0,10    12,16,30
0,11
0,12    3,10,16,29,30,38,41,55,83,85,89
0,13    27,37
0,14    4,19
0,15    4,27,80
0,16    10,12,18,30,38,89,53,83
0,17    19,26,28,53
0,18    4,16,30,89
0,19    14,17,50
0,2
```

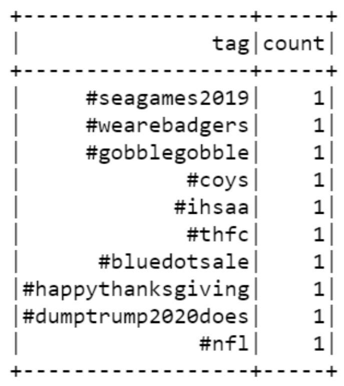
Question-2: Spark Data Frames:

RESULTS ARE SHOWN ABOVE

Question-3: Spark Streaming TaskPerform:



```
+-----------------+-----+
|              tag|count|
+-----------------+-----+
|    #seagames2019|    1|
|    #wearebadgers|    1|
|    #gobblegobble|    1|
|            #coys|    1|
|           #ihsaa|    1|
|            #thfc|    1|
|      #bluedotsale|    1|
|#happythanksgiving|    1|
|#dumptrump2020does|    1|
|             #nfl|    1|
+-----------------+-----+
```



```
+-----------------+-----+
|              tag|count|
+-----------------+-----+
|    #seagames2019|    1|
|    #wearebadgers|    1|
|    #gobblegobble|    1|
|            #coys|    1|
|           #ihsaa|    1|
|            #thfc|    1|
|      #bluedotsale|    1|
|#happythanksgiving|    1|
|#dumptrump2020does|    1|
|             #nfl|    1|
+-----------------+-----+
```

Question-4: Spark Graphx Task:

Page Rank Vertices.

In [13]:
```
pageRank.vertices.show()
```

```
+------+----------------+------------+------+--------+------------------+
|author|              id|       word2|source|sourceID|          pagerank|
+------+----------------+------------+------+--------+------------------+
|   312|       broadcast|transmission|    AC|       0|0.5327695560253642|
|  4522|          boring|    politics|    TF|       7|0.5327695560253642|
|  1317|           party|       house|   GOG|       4| 3.551797040169095|
|  6272|           party|        gala|    WP|       9| 3.551797040169095|
|  4240|           avoid|      devoid|   SAS|       6|0.5327695560253642|
|  4522|        bungalow|        flat|    TF|       7|0.5327695560253642|
|  4762|mocha choco latte| choco taco|    U2|       8|0.5327695560253642|
|   873|            sops|       gravy|   ECF|       3|0.5327695560253642|
|  2525|          blammo|         pow|   SAS|       6|0.5327695560253642|
|   387|         pickled|   cucumbers|    BC|       1|0.5327695560253642|
|    39|       criminals|       punks|    AC|       0| 3.551797040169095|
|   499|      englishman|        chap|    BC|       1|0.5327695560253642|
|  4633|            gaze|       watch|    TF|       7|0.5327695560253642|
|  2124|        disagree|    conflict|   GOG|       4|0.5327695560253642|
|  5713|          belief|       magic|    WP|       9|0.5327695560253642|
|  3925|           nurse|      degree|   SAS|       6|0.5327695560253642|
|  2779|      popularity|     popular|   SAS|       6|0.5327695560253642|
|  1089|          uproar|    brouhaha|   GOG|       4|0.5327695560253642|
|  2525|       forensics|         lab|   SAS|       6|0.5327695560253642|
|  1922|        vitamins|     oranges|   GOG|       4|0.5327695560253642|
+------+----------------+------------+------+--------+------------------+
only showing top 20 rows
```

Page Rank Edges.

In [14]:
```
pageRank.edges.show()
```

```
+----------+------------+------+------+
|       src|         dst|source|weight|
+----------+------------+------+------+
|flatulence|         gas|    TF|   1.0|
|        mr|          mr|    AC|   1.0|
|      tree|      leaves|    WP|   0.5|
|      tree|      leaves|    WP|   0.5|
|      tree|      leaves|    WP|   0.5|
|      tree|      leaves|    WP|   0.5|
|membership|        card|   GOG|   0.5|
|membership|        card|   GOG|   0.5|
|membership|        card|   GOG|   0.5|
|membership|        card|   GOG|   0.5|
|      edge|     science|    WP|   1.0|
|       tnt|    dynamite|   ECF|   1.0|
|      poke|        fire|    TF|   1.0|
|     shade|    coolness|    WP|   1.0|
|  airfield|        down|    WP|   1.0|
|    change|metamorphosis|    CC|   1.0|
|    apples|         red|    WP|   1.0|
|   barrier|       cream|   SAS|   1.0|
| shootings|    horrible|    TF|   1.0|
|       hat|       cover|   SAS|0.0625|
+----------+------------+------+------+
only showing top 20 rows
```

6. Conclusion

In this Lab Assignment we were able to revise ICP-1 to ICP-7 from module-2. This lab assignment made us to understand Hive, Solr and MapReduce concepts.