21 SEPT 2023 – ONLINE TV SERIES

MAPREDUCE & HADOOP

```
class Mapper1BigData extends Mapper<LongWritable, Text, Text, IntWritable> {

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

        String[] fields = value.toString().split(",");
        String sid = fields[0];
        String seasonNumber = fields[1];

        // Return the key-value pair
        // key = (sid, seasonNumber)
        // value = +1
        context.write(new Text(sid+","+seasonNumber), new IntWritable(1));
    }
}


class Reducer1BigData extends Reducer<Text, IntWritable, Text, IntWritable> {

    @Override
    protected void reduce(
            Text key,
            Iterable<IntWritable> values,
            Context context) throws IOException, InterruptedException {

        int numEpisodes = 0;

        for (IntWritable val : values) {
            numEpisodes += val.get();
        }

        String sid = key.toString().split(",")[0];

        // Return the key-value pair
        // key = sid
        // value = numEpisodes
        // The identifier of the season is not needed anymore
        context.write(new Text(sid), new IntWritable(numEpisodes));
    }
}
```

```
class Mapper2BigData extends Mapper<Text, Text, Text, Text> {

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

        int numEpisodes = Integer.parseInt(value.toString());

        // Return the key-value pair
        // key = sid
        // value = "S" if numEpisodes<=10, "L" otherwise
        if (numEpisodes <= 10)
            context.write(key, new Text("S"));
        else
            context.write(key, new Text("L"));
    }
}


class Reducer2BigData extends Reducer<Text, Text, Text, Text> {

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

        int longSeason = 0;
        int shortSeason = 0;

        for (Text value : values) {
            if (value.toString().compareTo("L") == 0) {
                longSeason++;
            } else {
                shortSeason++;
            }
        }

        context.write(key, new Text(longSeason + "," + shortSeason));
    }
}
```

SPARK

```
import pyspark

from pyspark import SparkContext
from pyspark.sql import SparkSession

spark = SparkSession.builder.getOrCreate()
sc = spark.sparkContext

custWatchedPath = "data/CustomerWatched.txt"
espisodesPath = "data/Episodes.txt"
tvSeriesPath = "data/TVSeries.txt"

# Useless for this program
# customersPath = "data/Customers.txt"

outputPath1 = "outPart1/"
outputPath2 = "outPart2/"

# cid, startTimestamp, sid, seasonN, epN
custWatchedRDD = sc.textFile(custWatchedPath)
# sid, seasonN, epN, title, OriginalAirDate
espisodesRDD = sc.textFile(espisodesPath).cache()
# sid, title, genre
tvSeriesRDD = sc.textFile(tvSeriesPath)

################################################################
# Part 1
################################################################

# Select only comedy TV series
# and map the result to pairs (SID, None) for the join with episodes
def filterComedy(l):
    genre = l.split(",")[2]
    return genre=='Comedy'

def mapSIDNone(l):
    SID = l.split(",")[0]
    return (SID, None)

comedyTVSeries = tvSeriesRDD.filter(filterComedy)\
                            .map(mapSIDNone)

# Map episodes to pairs:
```

```
# key = SID
# value = SeasonNumber

def mapSIDSeasonOne(l):
    fields = l.split(",")
    SID = fields[0]
    seasonNumber = fields[1]

    return (SID, seasonNumber)

episodesPairs = espisodesRDD.map(mapSIDSeasonOne)

# Join episodesPairs with comedyTVSeries to consider comedy TV series only
#
# Map to pairs
# - key = (SID, seasonNumber) - TV series season identifier
# - value = +1 - One more episode
# and use reduceByKey to count the number of episodes for each TV series season
TVseriesSeasonsNumEpisodes = episodesPairs.join(comedyTVSeries)\
                                    .map(lambda p: ( (p[0], p[1][0]), +1))\
                                    .reduceByKey(lambda v1, v2: v1+v2)

# Compute the average number of episodes per season for each comedy TV series
#
# Map to pairs:
# - key = SID
# - value = (number of episodes, +1)
#
# Sum the two parts of the values and then compute the average inside the
mapValues method

TVseriesAvgNumEpisodes = TVseriesSeasonsNumEpisodes.map(lambda p: (p[0][0],
(p[1], 1)))\
                                .reduceByKey(lambda p1, p2: (p1[0]+p2[0],
p1[1]+p2[1]))\
                                .mapValues(lambda p: p[0]/p[1])

# Store the result in the first output folder
TVseriesAvgNumEpisodes.saveAsTextFile(outputPath1)

############################################################
# Part 2
############################################################

# Compute the number of distinct seasons for each TV series
```

```
#
# Reuse episodesPairs (it contains one pair (SID, SeasonNumber) for each episode
# Apply distinct to consider each season only once per each TV series
# Then, map to pairs (SID, +1) and apply reduceByKey to compute the number of
distinct seasons for each TV series

NumSeasonsTvSeries = episodesPairs.distinct()\
                                  .map(lambda p: (p[0], 1))\
                                  .reduceByKey(lambda v1, v2: v1+v2)

# Compute for each combination (customer, TV series) the number of distinct
seasons of this TV series
# for which this customer watched at least one episode.
#
# Map each line of CustomerWatched.txt to pairs:
# - key = (customer, SID)
# - value = SeasonNumber
#
# Apply distinct to consider each TV series season only one time for each
customer
#
# Map the value part to +1 and then apply reduceByKey to count the number of
distinct seasons of this TV series
# for which this customer watched at least one episode.
def mapCustSIDSeasonNum(l):
    fields = l.split(",")
    cid = fields[0]
    sid = fields[2]
    seasonNumber = fields[3]

    return ( (cid, sid), seasonNumber)


CustomerTVSeriesNumSeasonsAtLeastOneVisualization =
custWatchedRDD.map(mapCustSIDSeasonNum)\
                                                              .distinct()\
                                                              .mapValues(lambda v:
+1)\
                                                              .reduceByKey(lambda
v1, v2: v1+v2)

# Map CustomerTVSeriesNumSeasonsAtLeastOneVisualization to pairs:
# - key = (SID TV Series, Number of distinct seasons of this TV series for which
this customer watched at least one episode)
# - value = CID Customer
```

```
SIDNumSeasonsCustomer = CustomerTVSeriesNumSeasonsAtLeastOneVisualization\
                            .map(lambda p: ( (p[0][1], p[1]), p[0][0]) )

# Map NumSeasonsTvSeries to pairs:
# - key = (SID TV Series, Number of distinct seasons of this TV series)
# - value = None

TvSeriesNumSeasonsNone = NumSeasonsTvSeries\
                            .map(lambda p: ( p, None))

# Join SIDNumSeasonsCustomer with TvSeriesNumSeasonsNone
# This natural join keeps the combinations (TV series, customers) such that the
number of seasons of the TV series
# is equal to the number of seasons of this TV series for which the customer
whatched at least one episode.
#
# Finally, use map to extract the selected combinations (SID, CID)

res2 = SIDNumSeasonsCustomer.join(TvSeriesNumSeasonsNone)\
                            .map(lambda p: (p[0][0], p[1][0]) )

# Store the result in the second output folder
res2.saveAsTextFile(outputPath2)
```