21 FEB 2022

HADOOP & MAPREDUCE

```
class MapperBigData extends Mapper<LongWritable, // Input key type
        Text, // Input value type
        IntWritable, // Output key type
        IntWritable> {// Output value type

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

            String[] fields = value.toString().split(",");
            Integer year = Integer.parseInt(fields[0].split("/")[0]);

            // Emit (year, +1))
            context.write(new IntWritable(year), new IntWritable(1));
    }
}


class ReducerBigData extends Reducer<IntWritable, // Input key type
        IntWritable, // Input value type
        IntWritable, // Output key type
        IntWritable> { // Output value type

    private int maxCount;
    private int maxYear;

    @Override
    protected void setup(Context context) throws IOException,
InterruptedException {
        maxCount = -1;
        maxYear = Integer.MAX_VALUE;
    }

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

        int numOccurrences = 0;
        int currentYear = key.get();
```

```java
        // Compute the number of occurrences of the current year
        for (IntWritable value : values) {
            numOccurrences = numOccurrences + value.get();
        }

        // Check if this is the local maximum
        if (maxCount < numOccurrences || (maxCount == numOccurrences &&
currentYear < maxYear)) {
            maxCount = numOccurrences;
            maxYear = currentYear;
        }
    }

    @Override
    protected void cleanup(Context context) throws IOException,
InterruptedException {
        // Emit the local maximum and the associated year
        if (maxCount != -1)
            context.write(new IntWritable(maxCount), new IntWritable(maxYear));
    }
}


class Mapper2BigData extends Mapper<Text, // Input key type
        Text, // Input value type
        NullWritable, // Output key type
        Text> {// Output value type

    // This is an identity mapper.
    protected void map(Text key, // Input key type
            Text value, // Input value type
            Context context) throws IOException, InterruptedException {

        String occurrences = key.toString();
        String year = value.toString();

        // Emit
        // key: NullWritbale
        // value: year + number of occurrences
        context.write(NullWritable.get(), new Text(year + "_" + occurrences));
    }

}


class Reducer2BigData extends Reducer<NullWritable, // Input key type
```

```
        Text, // Input value type
        IntWritable, // Output key type
        IntWritable> { // Output value type

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

        int maxCount = -1;
        int maxYear = Integer.MAX_VALUE;

        int currentYear;
        int numOccurrences;

        for (Text val : values) {
            String[] fields = val.toString().split("_");
            currentYear = Integer.parseInt(fields[0]);
            numOccurrences = Integer.parseInt(fields[1]);

            // Check if this is the glabal maximum
            if (maxCount < numOccurrences || (maxCount == numOccurrences &&
currentYear < maxYear)) {
                maxCount = numOccurrences;
                maxYear = currentYear;
            }
        }

        // Store the result
        context.write(new IntWritable(maxYear), new IntWritable(maxCount));
    }
}
```

SPARK

```
# customersInputPath = "data/Customers.txt" # Useless for this program
purchaseInputPath = "data/Purchases.txt"
catalogInputPath = "data/ItemsCatalog.txt"

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

# Define the rdds associated with the input files

# Input format: Timestamp,Username,itemID,SalePrice
purchaseRDD = sc.textFile(purchaseInputPath)

# Input format: itemID,Name,Category,Timestamp
catalogRDD = sc.textFile(catalogInputPath)

#########################################
# PART 1
#########################################

# Filter only the years 2020 and 2021 and cache this RDD, so that it can be
# reused for the second part
purchase20_21RDD = purchaseRDD\
                    .filter(lambda line: line.startswith("2020/") or
line.startswith("2021/"))\
                    .cache()

# Map to a pairRDD with:
# key = itemId
# value = (2020: 0/1, 2021: 0/1)

def mapItemID20202021(line):
    fields = line.split(",")
    year = fields[0].split("/")[0]
    itemId = fields[2]

    if (year=="2020"):
        return (itemId, (1, 0))
    else:
        return (itemId, (0, 1))


itemsYearsNumPurchsRDD = purchase20_21RDD.map(mapItemID20202021)
```

```
# Sum the number of purchases for each item in each of the two years
# key = itemId
# value = (num. purchases in 2020, num. purchases in 2021)
# Finally, select the items with num. purchases in 2020>=10000 and num. purchases
in
# 2021>=10000
validItems = itemsYearsNumPurchsRDD\
            .reduceByKey(lambda i1, i2: (i1[0]+i2[0], i1[1]+i2[1]))\
            .filter(lambda pair: pair[1][0] >= 10000 and pair[1][1] >= 10000)

# Store the selected items in the first output folder
# Store only the itemIDs
resPart1 = validItems.keys()

resPart1.saveAsTextFile(outputPath1)

##########################################
# PART 2
##########################################

# Start from previously cached RDD
# and consider only purchases made in 2020
# Map the considered pairRDD to a new PairRDD with
# key = (itemId, month)
# value = userId
# and perform a distinct operation to consider purchases made in each month of
# 2020 by distinct users.

def mapItemIdMonthUserID(line):
    fields = line.split(",")
    month = fields[0].split("/")[1]
    userId = fields[1]
    itemId = fields[2]

    return ((itemId, month), userId)



distinctPurchasesPerMoth2020RDD = purchase20_21RDD\
        .filter(lambda line: line.startswith("2020"))\
        .map(mapItemIdMonthUserID)\
        .distinct()

# Count the number of distinct customers for each item+month
# by first mapping the input pairs into the following pairs:
```

```
# key = (itemId, month)
# value = +1
# and then use a reduceByKey to sum the values.
# Finally, filter only those months for which the distinct customers were >= 10
itemsMonthsMoreThan9 = distinctPurchasesPerMoth2020RDD\
                    .mapValues(lambda v: 1)\
                    .reduceByKey(lambda i1, i2: i1 + i2)\
                    .filter(lambda it: it[1] >= 10)

# Count the number of months in 2020 in which each itemId was bought by >= 10
# distinct users
#
# Map to:
# key = itemId
# value = +1
#
# Use reduceByKey to count for each item the number of months of 2020 for which
# the number of distinct customers was >= 10
itemsNumMonthsCustomersMoreThan9 = itemsMonthsMoreThan9\
            .map(lambda it: (it[0][0], 1))\
            .reduceByKey(lambda i1, i2: i1 + i2)

# Select the items with more than 10 months with number of distinct customers
>=10
# These items must be discarded because they have at least 11 months each one
with
# num. distinct customers >= 10. In other words, they have at most one month
# with less than 10 distinct customers.
itemsWithManyMounthsWithManyCustomers = itemsNumMonthsCustomersMoreThan9\
                                        .filter(lambda it: it[1] > 10)

# Filter only items which were inserted in catalog before 01/01/2020 and
# Map the catalogRDD into a pairRDD with
# key = itemId
# value = Category

def mapItemIdCategory(line):
    fields = line.split(",")
    itemId = fields[0]
    category = fields[2]

    return (itemId, category)

itemCategoryRDD = catalogRDD\
            .filter(lambda line: line.split(",")[3]<"2020/01/01")\
            .map(mapItemIdCategory)

# Select the items occurring in itemCategoryRDD but not in
# itemsWithManyMounthsWithManyCustomers
# We need to use this approach in order to consider for each item also the
# months without sales (i.e., without customers).
# A month without sales has less than 10 distinct customers.
resPart2 = itemCategoryRDD.subtractByKey(itemsWithManyMounthsWithManyCustomers)

# Store the result in the second output folder
resPart2.saveAsTextFile(outputPath2)
```