20 JUNE 2024 – POLIJOB

MAPREDUCE & HADOOP

```
package it.polito.bigdata.hadoop;

import java.io.DataInput;
import java.io.DataOutput;
import java.io.IOException;

public class OfferStatusCounter implements org.apache.hadoop.io.Writable {

    private int accepted;
    private int rejected;

    public int getAccepted() {
        return accepted;
    }

    public int getRejected() {
        return rejected;
    }

    public void setAccepted(int accepted) {
        this.accepted = accepted;
    }

    public void setRejected(int rejected) {
        this.rejected = rejected;
    }

    @Override
    public void readFields(DataInput in) throws IOException {
        accepted = in.readInt();
        rejected = in.readInt();
    }

    @Override
    public void write(DataOutput out) throws IOException {
        out.writeInt(accepted);
        out.writeInt(rejected);

    }

}
```

```java
/*
Process JobOffers.txt to count
-  rejected offers with a salary higher than 100,000 euros --> these must be more
than 10
-  accepted offers --> these must be zero
Emits (jobID, [rejectedCount,acceptedCount]) to count in the reducer.
 */

public class Mapper1 extends Mapper<LongWritable, Text, Text, OfferStatusCounter>
{

    private final static int salaryThreshold = 100 * 1000;

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

        String[] fields = value.toString().split(",");

        // optional: skip the header (if any)
        if (fields[0].equals("OfferID")) {
            return;
        }

        String jobId = fields[1];
        double salary = Double.parseDouble(fields[2]);
        String status = fields[3];

        OfferStatusCounter offerStatusCounter = new OfferStatusCounter();

        // counting accepted offers
        if (status.equals("Accepted")) {
            // System.out.println("Mapper 1 emitting: " + jobId + " accepted");
            offerStatusCounter.setAccepted(1);
            context.write(
                new Text(jobId),
                offerStatusCounter);
        }
        // counting rejected offers with salary > 100k euros
        else if (status.equals("Rejected") && salary > salaryThreshold) {
            // System.out.println("Mapper 1 emitting: " + jobId + " rejected");
            offerStatusCounter.setRejected(1);
            context.write(
                new Text(jobId),
                offerStatusCounter);
```

```
        }
    }
}


/**
 * Reducer1 - Counting rejected and accepted offers per job ID
 * Receives as input:
 * (jobID, [1,0]) or
 * (jobID, [0,1])
 * with an offerStatusCounter object that counts the number of rejected and
accepted offers.
 */
class Reducer1 extends Reducer<
                Text,           // Input key type
                OfferStatusCounter,    // Input value type
                Text,           // Output key type
                Text> {  // Output value type

    @Override

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

        // Iterate over the set of values and sum them
        int sumAccepted = 0;
        int sumRejected = 0;
        for (OfferStatusCounter value : values) {
            sumAccepted += value.getAccepted();
            sumRejected += value.getRejected();
        }

        if (sumAccepted == 0 && sumRejected >= 10) {
            context.write(
                key,                        // Job ID
                new Text(""+sumRejected)); // Number of high-salary rejected
offers
        }
    }
}
```

SPARK

```
from pyspark import SparkConf, SparkContext

conf = SparkConf().setAppName('Exam 20 june 2024')
sc = SparkContext(conf = conf)

jobContractsPath = "sample_data/JobContracts.txt"
jobOffersPath = "sample_data/JobOffers.txt"
jobPostingsPath = "sample_data/JobPostings.txt"

outputPath1 = "outSpark1/"
outputPath2 = "outSpark2/"

# JobID,Title,Country
jobPostingsRDD = sc.textFile( jobPostingsPath )
# OfferID,JobID,Salary,Status,SSN
jobOffersRDD = sc.textFile( jobOffersPath)
# ContractID,OfferID,ContractDate,ContractType
jobContractsRDD = sc.textFile( jobContractsPath )

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

# Filter accepted job offers and extract (JobID,(Salary, OfferId))
# OfferId is helpful for solving the second part.
# To avoid repeating the same join in the second part, we already retrieve the
OfferId here.

def jobIDSalaryOfferID(line):
    fields = line.split(",")

    offerId = fields[0]
    jobID = fields[1]
    salary = float(fields[2])

    return (jobID, (salary,offerId))


acceptedOffers = jobOffersRDD\
    .filter(lambda line: line.find(",Accepted,")>=0)\
    .map(jobIDSalaryOfferID)

# Extract (JobID, (Country, Title)) from job postings
```

```
# Title is helpful for solving the second part.
# To avoid repeating the same join in the second part, we already retrieve the
title here.

def jobIDCountryTitle(line):
    fields = line.split(",")

    jobID = fields[0]
    title = fields[1]
    country = fields[2]

    return (jobID, (country,title))

jobIDCountry = jobPostingsRDD.map(jobIDCountryTitle)

# Join accepted offers with job postings to get (JobID, ((Salary, OfferId),
(Country, Title)))
offersWithCountry = acceptedOffers.join(jobIDCountry).cache()

# Map to (Country, (Salary, 1))
countrySalaryCount = offersWithCountry\
    .map(lambda tuple: (tuple[1][1][0], (tuple[1][0][0], 1)))

# Reduce by key to get (Country, (TotalSalary, Count)) -> a = TotalSalary, b =
Count
countryTotalSalaryCount = countrySalaryCount\
    .reduceByKey(lambda a, b: (a[0] + b[0], a[1] + b[1]))

# Map to (Country, AverageSalary)
countryAverageSalary = countryTotalSalaryCount\
    .mapValues(lambda tuple: float(tuple[0]) / float(tuple[1]))

# Select only the top N countries with the highest average salary
topCountries = countryAverageSalary\
    .top(3, lambda tuple: tuple[1])

# Convert the top 3 countries to an RDD and save the result in the output folder
topCountriesRDD = sc.parallelize(topCountries)

topCountriesRDD.saveAsTextFile(outputPath1)


######################
# PART 2
######################
```

```
# Map offersWithCountry to
# (OfferID, (Country, Title))

def offIdTitleCountry(pair):
    offerId = pair[1][0][1]
    country = pair[1][1][0]
    title = pair[1][1][1]

    return (offerId, (title, country))


offerTitleCountry = offersWithCountry.map(offIdTitleCountry)

# Map contracts to (OfferID, None)

def offIdNone(line):
    # ContractID,OfferID,ContractDate,ContractType
    fields = line.split(",")
    offerID = fields[1]

    return (offerID, None)


offerIdContract = jobContractsRDD.map(offIdNone)

# Join offerTitleCountry with offerIdContract -> (offerId, ((Title, Country),
null))
# map to ((title, country), +1)
# and apply reduceByKey to count the number of contracts for each title in each
country
# ((title, country), numContracts)


titleCountryNumContracts = offerTitleCountry.join(offerIdContract)\
    .map(lambda pair: (pair[1][0], 1))\
    .reduceByKey(lambda v1, v2: v1+v2).cache()

# Map to (Country, numContracts) and compute the maximum for each country
def CountryNumContracts(pair):
    country =  pair[0][1]
    numContracts = pair[1]

    return (country, numContracts)
```

```
countryMaxNumContracts = titleCountryNumContracts\
    .map(CountryNumContracts)\
    .reduceByKey(lambda v1, v2: max(v1,v2))

# Map countryMaxNumContracts to
# ( (Country, maxNumContracts), None )
countryMaxNull = countryMaxNumContracts.map(lambda pair: (pair, None))


# Map Join titleCountryNumContracts to ((country, numContracts), title)

def CountryNumContractsTitle(pair):
    title = pair[0][0]
    country = pair[0][1]
    numContracts = pair[1]

    return ((country,numContracts), title)


countryNumContractsTitle = titleCountryNumContracts.map(CountryNumContractsTitle)

# Join countryNumContractsTitle with countryMaxNull
# and map to the string Country,Title,NumberOfContracts
mostPopularTitlePerCountry = countryNumContractsTitle.join(countryMaxNull)\
    .map(lambda pair: pair[0][0]+ "," + pair[1][0] + "," + str(pair[0][1]))

# Save the result to the output folder
mostPopularTitlePerCountry.saveAsTextFile(outputPath2)
```