Exam 20 june 2024

The managers of PoliJob, an international job portal company, asked you to develop two applications:

- a MapReduce program (Exercise 1)
- a Spark-based program (Exercise 2)

to address the analyses they are interested in. The applications can read data from three input files: `JobPostings.txt`, `JobOffers.txt`, and `JobContracts.txt`.

JobPostings.txt

- It is a textual file containing information about the job postings, i.e., positions for jobs that companies are looking for. There is one line for each job posting. The `JobID` uniquely identifies the job postings. Multiple job postings with the same position (title) can be present. This file is extremely large and you cannot suppose its content can be stored in one in-memory Java variable.
- Each line has the following format:
  - `JobID,Title,Country`
- Example data:
  - `1,Software Engineer,IT`
  - `2,Data Scientist,FR`
  - `3,Software Engineer,ES`
  - `4,Software Engineer,IT`
  - `5,Data Scientist,IT`
- Example: The Job 1 (`JobID`) offers the position (title) of Software Engineer in Italy (`IT`).

JobOffers.txt

- It is a textual file containing information about the job offers, i.e., job offers that were proposed to candidates applying for a Job posting (`JobID`). There is one line for each offer. `OfferID` uniquely identifies the offers. The status indicates whether the offer has been accepted or not. If an offer is accepted, then you might have a job contract (i.e., there can be from 0 to 1 job contract for each offer in `JobContracts.txt`). This file is extremely large and you cannot suppose its content can be stored in one in-memory Java variable.
- Each line has the following format:
  - `OfferID,JobID,Salary,Status,SSN`
- Example data:
  - `101,1,97000,Accepted,800-11-2222`
  - `102,2,120000,Rejected,800-11-2223`
  - `103,3,120000,Accepted,800-12-2222`
  - `104,5,120000,Rejected,801-21-3222`
  - `105,5,120000,Rejected,800-41-2232`
  - `106,4,120000,Rejected,800-14-2422`
  - `107,3,120000,Rejected,800-17-2252`
  - `114,5,120000,Accepted,800-51-2222`
  - `115,1,120000,Accepted,800-51-2622`

    - `116,5,120000,Accepted,800-14-2262`
    - `117,5,120000,Accepted,800-14-2262`
- Example: The offer 101 has been proposed to a candidate for job 1 (`JobID`) with a salary of 97 thousand euros, and the candidate accepted the offer. `800-11-2222` (SSN) is the social security number of the candidate to whom the offer has been made.

JobContracts.txt

- It is a textual file containing information about the job contracts signed by candidates and companies, i.e., after an offer has been accepted, a contract is typically signed (there are from 0 to 1 contract for each accepted offer). There is one line for each job contract, and `ContractID` uniquely identifies the contracts. This file is extremely large and you cannot suppose its content can be stored in one in-memory Java variable.
- Each line has the following format:
    - `ContractID,OfferID,ContractDate,ContractType`
- Example data:
    - `201,101,2023-01-15,Full-time`
    - `202,103,2023-02-01,Part-time`
    - `203,114,2023-03-10,Full-time`
    - `204,115,2023-04-20,Internship`
    - `205,116,2023-05-05,Full-time`
    - `206,117,2023-06-15,Contractor`
- Example: The contract 201 for the offer 101 has been signed on Jan 15, 2023, with type “Full-time”.

Map reduce exercise [... to do]

Exercise 2 – Spark and RDDs (19 points)

The managers of PoliJob asked you to develop a single Spark-based application based on RDDs or Spark SQL to address the following tasks. The application takes the paths of the three input files, and two output folders (associated with the outputs of the following points 1 and 2, respectively).

1. **Top 3 countries with the highest average salary**:

    - For each country, compute the average salary, considering only accepted job offers.
    - Select the top 3 countries with the highest value of average salary.
    - The first HDFS output file must contain the identifiers of the selected countries (one country per output line) and their average salary (computed considering only accepted job offers).
    - **Note**: Suppose there is at least an accepted job offer for each country.

2. **Select the most popular job title in each country among job postings that resulted in job contracts**:

    - Identify the job title with the highest number of job contracts in a given country.
    - Store in the output folder the country names, their most common job title(s), and the contract count for that job title(s).

- **Note**: For each country, there might be many job titles associated with the highest number of job contracts for that country. Store all of them in the output folder (one of them per output line together with country and the contract count).
- **Note**: Suppose there is at least a contract for each country.

Solution Approach

Part 1 - Top 3 Countries with Highest Average Salary

Steps:

1. **Filter Accepted Offers**:
   - Extract only the accepted job offers from the `JobOffers.txt` file.
   - Map each accepted offer to a pair: `(JobID, (Salary, OfferID))`.
2. **Join Job Postings with Accepted Offers**:
   - Join the filtered accepted offers with the `JobPostings.txt` RDD based on `JobID`.
   - This gives us a pair of `JobID` and a tuple of `((Salary, OfferID), (Country, Title))`.
3. **Map to Country-Salary Pair**:
   - Map the result to `(Country, (Salary, 1))`, where `1` represents the count of accepted offers for that job title in that country.
4. **Reduce by Key (Country)**:
   - Sum the total salary and count of offers for each country.
5. **Compute Average Salary**:
   - For each country, divide the total salary by the number of accepted offers to get the average salary.
6. **Top 3 Countries**:
   - Use the `.top(3, lambda tuple: tuple[1])` method to select the top 3 countries with the highest average salary.
7. **Save the Result**:
   - Save the top 3 countries with their average salary to the output folder `outSpark1/`.

Part 2 - Most Popular Job Title in Each Country Among Job Postings with Contracts

Steps:

1. **Map Offers with Title and Country**:
   - Map the `offersWithCountry` RDD to extract a pair: `(OfferID, (Title, Country))`.

2. **Map Job Contracts to OfferID**:
    - Extract the `OfferID` from the `JobContracts.txt` file, mapping it to `(OfferID, None)`.
3. **Join Offers with Job Contracts**:
    - Join the `offerTitleCountry` RDD with the `offerIdContract` RDD on `OfferID` to get `(OfferID, ((Title, Country), None))`.
4. **Map to (Country, (Title, 1))**:
    - For each valid contract, map the result to `(Country, (Title, 1))`, where `1` represents the number of contracts for that job title in that country.
5. **Count Contracts by Title and Country**:
    - Use `.reduceByKey` to count the number of contracts for each job title in each country.
6. **Find Maximum Contracts per Country**:
    - Map the result to `(Country, numContracts)` and apply `.reduceByKey` to find the maximum number of contracts for each country.
7. **Filter for Most Popular Titles**:
    - Map the result to `((Country, numContracts), Title)` to get a pair with the country, the number of contracts, and the corresponding job title.
8. **Join with Maximum Contracts**:
    - Join the job titles and contract counts with the maximum contracts per country to get only the most popular job titles.
9. **Save the Result**:
    - Save the results, including the country name, the most popular job title(s), and the number of contracts, to the output folder `outSpark2/`.

Code

```
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/"
```

```python
# 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)))
```

```python
# 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)
```