Data AI exam

The goal of the examination is to test your understanding of distributed frameworks and your capacity to manipulate them in an efficient way. This test will not be graded by the syntactic correctness of your code but rather by the demonstration that you can write (efficient) distributed code. For each question, you will be penalized if your answer uses Spark/Hadoop inefficiently (e.g. doing `collect` with too much data, shuffling sensibly more data than needed or doing a useless shuffle).

I need to understand what you are writing so please be clear, use the Python language (even for Map Reduce) and comment your code. You can use a mix of Python and Pseudo-code but be very clear on how this would translate to Python. Here are several examples of what I expect if it was asked to compute how many times each input items appears (classic word count).

**Pythonic style with RDD**

```python
input RDD is R

R .map(lambda x: (x,1))
 .reduceByKey(lambda a,b: a+b) #don't hesitate to break lines!
```

**Pythonic style with RDD without anonymous functions**

```python
def mapper(x):
    return (x,1)

def reducer(a,b):
    return a+b

R.map(mapper).reduceByKey(reducer)
```

**Pseudo-code + Python style with Spark**

```python
R.map(x → (x,1)).reduceByKey(a,b → a + b)
```

**Dataframe code**

```python
input.groupBy("item").count()
```

**Dataframe alternative**

```python
input.createOrReplaceTempView("input")
spark.sql("SELECT COUNT(*) FROM input GROUP BY item")
```

**Hadoop (provide only the mapper, reducer and eventually the combiner functions)**

```python
def mapper(x): # returns a list
    return [(x,1)]

def reducer(key, vals): # takes a key and a list of values
    return sum(vals) # returns an element or a list
```

*Final note:* please add the question number before each of your answer (e.g. 1.2 or 3.2).
1. Spark exercise: count number of elements under a threshold

1.1. Using the RDD interface
Given a threshold value $V$ and an RDD $R$ containing numeric values, count the number of elements $e$ in $R$ such that $e < V$.

1.2. Using the DataFrame interface
Given a threshold value $V$ and a DataFrame $D$ composed of one column $c$ containing numeric values, count the number of rows $r$ in $R$ such that the value of the column $c$ of row $r$ is strictly less than $V$ (i.e. $r[c] < V$).

1.3. Using the Map-Reduce interface
Given a threshold value $V$ and an HDFS file containing on each line a numeric value (represented as a string), count the number of line $r$ in $R$ such that the value $e$ represented by $r$ is strictly less than $V$ ($e < V$).

2. Root cause
You have a dataset $D$ containing a set of pairs $(i, c)$ where $i$ is the identifier of an object (i.e. an animal, a person, etc.) and $c$ corresponds to an attribute of $i$. For instance the dataset below describes that bats and horses are mammals while crows and bats can fly.

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Attribute</th>
</tr>
</thead>
<tbody>
<tr>
<td>crow</td>
<td>canFly</td>
</tr>
<tr>
<td>horse</td>
<td>isMammal</td>
</tr>
<tr>
<td>bat</td>
<td>isMammal</td>
</tr>
<tr>
<td>bat</td>
<td>canFly</td>
</tr>
</tbody>
</table>

An attribute $C'$ is salient for another attribute $C$ within a dataset $D$ when the proportion of objects having the attribute $C'$ among the set of objects having the attribute $C$ in $D$ is greater than the proportion of objects having the attribute $C'$ in the whole set $D$. For instance if $D$ is the dataset above and $C = \text{isMammal}$, we do not expect to see canFly as only half of our mammals can fly while two-third of all animals (in our dataset) can fly.

Your are given the dataset and an attribute and you want to compute its salient attributes. It should not have an impact on your code but you can suppose that a pair cannot appear twice. Furthermore we also suppose that the input attribute does appear in the dataset (otherwise the problem is ill-defined).

2.1. Using the RDD interface
Given an RDD $D$ containing pairs of strings describing a dataset and a attribute $C$ (given as a string), compute the salient attributes of $C$.

2.2. Using the DataFrame interface
Given an attribute $C$ (given as a string) and a dataset given as a DF $D$ containing two columns “identifier” and “attribute” containing strings describing a dataset, compute the salient attributes of $C$. 

Louis Jachiet 10/11/2021
3. Multiplying sparse matrices

A matrix is said sparse when it is mostly composed of zeros. While a matrix of dimension \( n \times m \) is typically represented as an array of \( n \times m \) numbers, if the matrix is sparse it can be represented more efficiently by giving the values only for non-zero positions. For instance, we can represent such a matrix as a set \( T \) of triples \((x, y, v)\) indicating that \( M_{x,y} = v \), and whenever there is a pair \((x, y)\) not included in \( T \) it is implied that \( M_{x,y} = 0 \). Obviously \( T \) is such that we don’t find twice the pair \((x, y)\).

You are given two sparse matrices: \( A \) of dimension \( h \times k \) and \( B \) of dimension \( k \times w \) and you have to compute their product (of dimension \( h \times w \)). Both \( A \) and \( B \) are given as RDDs containing triples.

You can assume that the dimensions \( h, k \) and \( w \) are less than \( 10^6 \) (which means that it is possible to group values by line or by column) but it is completely impossible to materialize the full matrix. To get full points, your algorithm should be correct & efficient!

4. The page-rank algorithm in Spark RDD

The web can be represented as a graph where vertices are webpages and there is an edge from page \( A \) to page \( B \) when page \( A \) has a link towards page \( B \). The page rank algorithm is an algorithm that takes as input a graph composed of vertices and edges and outputs a weight for each of the vertices. The page rank is named that way because it has been used to “rank” the pages of the web. This is very useful for search engines to retrieve among all the documents matching a query the ones that correspond to the most trustworthy pages.

One way to represent the page rank algorithm is to imagine a population of people randomly browsing the web. At each step, each person is on a given webpage. To compute the webpage visited at the next step we use the following probability distribution:

- With probability \( \alpha \), the next page is selected randomly across all webpages.
- With probability \( 1 - \alpha \), the next page is selected among the webpages pointed by the current webpage.

Note that this distribution only works if there is at least one webpage pointed by the current webpage. In the case that there is none, the next webpage is simply selected randomly among all webpages.

The score given by the page rank algorithm to a page is simply the probability to end up in this page starting from a random page after a large number of iterations. In this appendix a python code to compute this is provided.

For this exercise, we will suppose that we are provided with two RDDs:

- An RDD called webpages containing the identifiers of webpages.
- An RDD called pointsTo containing pairs \((i_1, i_2)\) meaning that the page identified by \( i_1 \) points to the page identified by \( i_2 \).

We will also suppose that there is a variable called alpha storing the parameter \( \alpha \) as well as a variable called nbIter storing the number of iterations to be done.

Your task, for this exercise, is to implement the page rank using Spark RDDs that returns an RDD called score containing pairs \((i, p)\) where \( i \) is the identifier of a webpage and \( p \) is the page rank score.

You are free to write the page rank as you wish (as long as it uses RDD) but it is recommended to follow these steps:

- Compute an RDD containing the equiprobability, that is pairs \((i, p)\) where \( i \) is the identifier of a page and \( p = \frac{1}{N} \) where \( N \) is the number of pages.
- Compute an RDD containing pairs \((i, n)\) where \( i \) is the identifier of a page and \( n \) is the number of pages pointed by the page identified by \( i \).
• Compute the probability of a random teleportation (similar to the variable `probability_teleportation` in the python code), given a score RDD (in the same format as the output).

• Compute the score after one iteration (i.e. given a score RDD compute the new score)

• Compute the complete page rank.

A. Python implementation of page rank

```python
# pointsTo is an array of list describing the page pointed
# nbPages is the number of pages
# alpha is a parameter of the algorithm
# nbIter is a second parameter of the algorithm

score = [ 1/nbPages for _ in range(nbPages) ]
for _ in range(nbIter):
    # we start with the new score at 0
    newScore = [ 0 for _ in range(nbPages) ]

    # then we add the probability of ending up in a page
    # after following a link
    for page in range(nbPages):
        for nxt in pointsTo[page]:
            new_score[nxt] += score[page] * (1-alpha) / len(pointsTo[page])

    # then we we compute the probability of moving to a random webpage
    probability_teleportation = 0
    for page in range(nbPages):
        if len(pointsTo[page]):
            probability_teleportation += alpha * score[page]
        else:
            probability_teleportation += score[page]

    # then we add to new_score the random moves
    for page in range(nbPages):
        new_score[page] += probability_teleportation / nbPages

    # finally we override the score array with the new scores
    score = new_score
```
### PySpark & Spark SQL

Spark SQL is Apache Spark's module for working with structured data.

---

### Initializing SparkSession

A SparkSession can be used to create DataFrames, register DataFrames as tables, execute SQL over tables, cache tables, and read parquet files.

```python
>>> from pyspark.sql import SparkSession
>>> spark = SparkSession.builder.getOrCreate()
```

### Creating DataFrames

From RDDS

```python
>>> from pyspark.sql.types import *
>>> sc = spark.sparkContext
>>> parts = sc.textFile("people.txt").map(lambda p: p.split(" "))
>>> people = parts.map(lambda p: Row(name=p[0], age=int(p[1])))
>>> peopledf = spark.createDataFrame(people)
```

### From Spark Data Sources

```python
>>> df = spark.read.json("customer.json")
```

### JSON

```python
>>> df = spark.read.json("customer.json")
```

### DataFrames

```python
>>> df = spark.read.csv("people.txt", inferSchema=True)
```

### Parquet files

```python
>>> df2 = spark.read.parquet("data.parquet")
```

### TXT files

```python
>>> df3 = spark.read.text("users.parquet")
```

---

### Duplicate Values

```python
>>> df = df.dropDuplicates()
```

---

### Queries

```python
>>> from pyspark.sql import functions as F
>>> df.select("firstName", "lastName")
>>> df.withColumn("age", F.\(\text{explode}(\text{phoneNumber})\))
```

### Show all entries in firstName column

```python
Show all entries in firstName, age and type
```

### GroupBy

```python
>>> df.groupBy("age")
```

### Filter

```python
>>> df.filter(\(\text{df["age"]}>24\))
```

### Sort

```python
>>> df.sort("age", ascending=False).collect()
```

### Creating DataFrames

From RDDs

```python
>>> people = parts.map(lambda p: Row(name=p[0], age=int(p[1])))
>>> peopledf = spark.createDataFrame(people)
```

### Add, Update & Remove Columns

Adding Columns

```python
>>> df = df.withColumn("city", df.address.city)
```

### Display all entries in telephoneNumber column

```python
>>> df.select("telephoneNumber")
```

### Updating Columns

```python
>>> df = df.withColumnRenamed("telephoneNumber", "phoneNumbers")
```

### Removing Columns

```python
>>> df = df.drop("address", "phoneNumber")
```

---

### GroupBy

```python
>>> df.groupBy("age").count() \.show()
```

### Filter

```python
>>> df.filter(df["age"]>24).show()
```

### Sort

```python
>>> df.sort("age", ascending=False).collect()
```

### Group by age, count the members in the groups

```python
>>> df.groupBy("age").count().show()
```

### Filter entries of age, only keep those records of which the values are >24

```python
>>> df.filter(df["age"]>24).show()
```

### Replace null values

Return new df replacing one value with another

```python
>>> df = df.na.fill(50)
```

### Running SQL Queries Programmatically

Registering DataFrames as Views

```python
>>> peopledf.createTempView("people")
```

### Query Views

```python
>>> df5 = spark.sql("SELECT * FROM customer")
```

---

### Output

Data Structures

```python
>>> df1.toPandas()
```

### Write & Save to Files

```python
>>> df1.save("namesAndAges.json", format="json")
```

---

### Stopping SparkSession

```python
>>> spark.stop()
```
Spark RDD

Spark operators are either lazy transformation transforming RDDs or actions triggering the computation.

**Import/Export**

- `myRDD = textFile(f)` Read f into RDD
- `myRDD.saveAsTextFile(f)` Store RDD into file f.
- `myRDD = sc.parallelize(l)` Transform list l into RDD.

**Transformations on one RDD without a shuffle**

These functions transform RDDs into other RDDs. All these functions are lazy and do not imply a shuffle.

- `myRDD.filter(f)` Keep rows r where f(r) is True
- `myRDD.map(f)` Transform each row r into the row f(r).
- `myRDD.flatMap(f)` Transform each row into the set of rows f(r).
- `myRDD.mapValues(f)` Transform each row (k, v) into the row (k, f(v)).
- `myRDD.flatMapValues(f)` Transform each row (k, v) into the set of rows (k, f(v)).
- `myRDD.keyBy(f)` Transform each row r into the row (f(r), r).
- `myRDD.distinct()` Return an RDD which is a subset of the RDD. Expect rows to be pairs!
- `myRDD.sample(replacement, fraction, seed)` Return a sample of the RDD. The parameter `replacement` controls whether an element can be sampled more than once, `fraction` controls the expected number of times an element appears (you should have $0 \leq \text{fraction} \leq 1$) and `seed` is optional and controls the random generator.

**Transformations on one RDD with a shuffle**

These functions transform RDDs into other RDDs with a shuffle. All the data is shuffled.

- `myRDD.distinct()` Return an RDD with the set of values but without duplicates.
- `myRDD.groupByKey()` Group all the values associated with a key. The result is an RDD containing pairs of a key and a list of all the values associated with this key. All the data is shuffled.
- `myRDD.reduceByKey(f)` Group all values associated with a key. As long as there is (k, v1) and (k, v2) in the RDD, they are replaced with k, f(v1, v2) until each key is unique in the RDD. Only one value per partition is shuffled.
- `myRDD.foldByKey(zero, f)` Group all values associated with a key. Let v0, ..., vk be the values associated with k in a partition, the function computes f(v0, ..., f(vk, zero)...) then let p1, ..., pk be the values associated with k in the various partition it computes r(p1, ..., r(p0, p1)...). Only one value per partition is shuffled.
- `myRDD.reduceByKey(f)` Group all values associated with a key. Let v0, ..., vk be the values associated with k in a partition, the function computes f(v0, ..., f(vk, zero)...) then let p1, ..., pk be the values associated with k in the various partition it computes r(p1, ..., r(p0, p1)...). Only one value per partition is shuffled.

**Miscellaneous**

- `myRDD.sort()` Sort the RDD according to the value returned by f.
- `myRDD.sortWith(f)` Sort the RDD according to the value returned by f. Ensure that the RDD is cached in RAM.
- `myRDD.persist()` Ensure that the RDD is cached in RAM.
- `myRDD.unpersist()` Ask Spark to free the memory of the given RDD.
- `myRDD.persist()` Ask Spark to free the memory of the given RDD.

**Quick SQL recall**

```
SELECT col1, ..., colK, sum(colE), min(colE) FROM table1 t1, ..., tableK tk
WHERE condition GROUP BY colA, colB
```

Condition can be:

- `conditionA AND conditionB`
- `conditionA OR conditionB`
- `NOT condition`
- `EXISTS (SELECT * FROM ...)`