Assignments 6

Using Hadoop MapReduce, create MapReduce scripts that process data provided by datasets and text files.

Pre-Work: Run the **$ pip install mrjob** command so it is possible to run this assignment.

Task#1: Average Number of Friends by Age.

$ python task1_lab6.py fakefriends.csv

Start by creating the mapper and reduce functions needed

1. Mapper function:
    a. Extract all lines of ‘fakefriends.csv’ and load them into 4 variables that represent each value of a line of the dataset.
    b. yield the age and the number of friends of each line.
2. Reducer function:
    a. In each age value, sum the number of friends and count the number of people that have that age.
    b. Divide those two values to calculate the average of friends at a given age.
    c. yield age and the average of friends per age.

3. Step function:
    a. Define a single MRStep with the mapper and reducer discussed before.

Task#2: Minimum Temperature by Capital

$ python task2_lab6.py 1800.csv

1. Start by creating the mapper and reduce functions needed
2. Mapper function:
    a. Extract all lines of ‘fakefriends.csv’ and load into 2 variables the values present in the first and fourth column of the dataset, these being the only values we are going to use, the weather station and temperature value.
    b. yield these values.
3. Reducer function:
    a. In each weather station, get the minimum temperature registered.
    b. yield the station and the value.
    c. In truth we could have used the type of observation field, to limit the values we need to search through, but since the performance of the

script was good in this case, we decided to go with the easier version of this script.

4. Step function:
    a. Define a single MRStep with the mapper and reducer discussed before.

Task 3: Sort the Word Frequency in a Book.

$ python task3_lab6.py Book.txt

Start by creating the *mapper* and *reduce* functions needed:

1. Mapper function:
    a. Extract all lines of 'Book.txt' and get all words based on a simple regex.
    b. yield each word and the number 1 as its count value.
2. Reducer function:
    a. In each word, sum its count value, being that each occurrence of a given word count 1, so the sum adds up to the correct value.
    b. yield the word and the count of occurrences.
3. Mapper sort function:
    a. Swap the order of the word and total, in the yield, so we can sort based on the count value and not the word.
    b. Do the '%04d' % int(total) operation so that the amount spent has enough left padding to be correctly sorted. Being that in MapReduce jobs the sort is lexicographical, this means that 4 is bigger that 41 for example, but by adding the padding we get 0004 and 0041, giving us that 0041 is bigger than 0004 as intended.
    c. yield the count and word values.
4. Reducer sort function:
    a. For each word, yield the word and the respective count of occurrences.
5. Step function:
    a. Define a step with two MRSteps, being the first with the mapper and reducer and the second one with the sorter mapper and reducer, all being discussed before.

Task 4: Sort the Total Amount Spent by Customer.

$ python task4_lab6.py customer-orders.csv

Start by creating the mapper and reduce functions needed:

1. Mapper function:
    a. Extract all lines of 'customer-orders.csv'' and get its value of customerId and the amount, all based on a simple regex.
    b. yield both values.
2. Reducer function:
    a. For each customer, sum the amount spent by him.

   b. yield the customerId and the sum spent.
3. Mapper sort function:
   a. Swap the order of the customerId and amount spent in the yield so we can sort based on the count value and not the word, as we did in the third task.
   b. Add padding to the number because of the same reason shown in the third task, being that this time we also want the two decimal places to be considered.
4. Reducer sort function:
   a. For each customer, yield its id and total amount spent.
5. Step function:
   a. Define a step with two MRSteps, the first with the mapper and reducer and the second one with the sorter mapper and reducer, all being discussed before.

To run all the scripts from the following Assignments, we use the command **spark-submit** on our terminal.

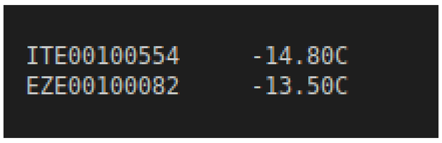
Assignment 7

In this task, using Spark RDDs, we apply the following operations:

Task 1: Find the minimum temperature per city

1. filter - to only select the row containing the indicated type of information ('TMIN')
2. map - to transform the single string with all the information to the different columns, by splitting by spaces
3. map - to transform the temperature's type of attribute from string to integer, for it to be compared and to find the minimum as requested.
4. reduceByKey - to apply the minimum function to the rows that have the same key, hence finding the minimum value for each station.

Output:

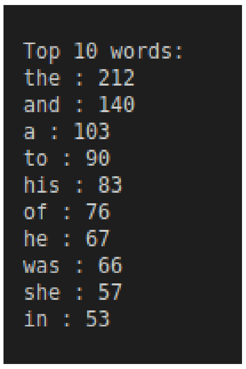

```
ITE00100554     -14.80C
EZE00100082     -13.50C
```

Tasks 2 and 3: Obtain and sort the word frequency in a Book

1. flatmap - to separate the words in rows instead of all being inside the same value
2. map - to transforme each word in a tuple containing the word and a counter, beginning in 1.
3. reduceByKey - to sum the counters of the rows with the same key, that is the word itself.

4. sortBy - to sort the RDD by the occurrence of each word in descending order, for the most frequent words to appear on the top **(Task 3)**

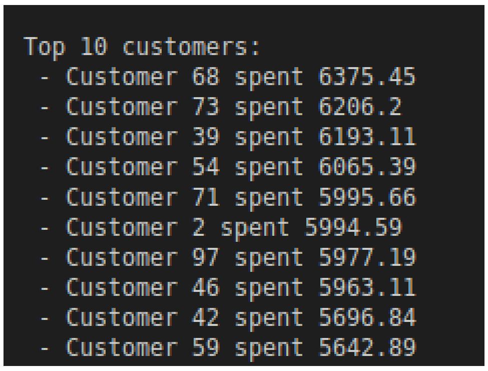
Output:


```
Top 10 words:
the : 212
and : 140
a : 103
to : 90
his : 83
of : 76
he : 67
was : 66
she : 57
in : 53
```

Tasks 4 and 5: Obtain and sort the total amount spent by customer

1. map - split the lines using commas
2. map - store the customerID and cast and store the amount spent on each product
3. reduceByKey - to sum the amount spent for every row that have the same key (customerID)
4. sortBy - to sort the RDD in descending order, for the customers that spent the most to appear on the top **(Task 5)**

Output:


```
Top 10 customers:
 - Customer 68 spent 6375.45
 - Customer 73 spent 6206.2
 - Customer 39 spent 6193.11
 - Customer 54 spent 6065.39
 - Customer 71 spent 5995.66
 - Customer 2 spent 5994.59
 - Customer 97 spent 5977.19
 - Customer 46 spent 5963.11
 - Customer 42 spent 5696.84
 - Customer 59 spent 5642.89
```

Tasks 6 and 7: Find the most and least popular superhero in the dataset

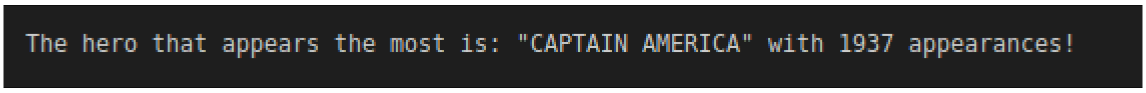
Movies RDD:

1. flatmap - to separate the numbers from each line in separated rows instead of all being in the same line
2. filter - to remove empty strings from the splitting process
3. map - to transforme each word in a tuple containing the hero ID and a counter, beginning in 1.
4. reduceByKey - to sum the counters of the rows with the same key, that is the hero ID itself.
5. sortBy - to sort the RDD by the occurrence of each hero in descending order, for the most frequent heros to appear on the top **(Task 6)**
6. sortBy - to sort the RDD by the occurrence of each hero in ascending order, for the least frequent heros to appear on the top **(Task 7)**

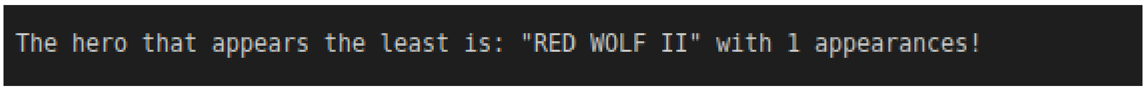
Hero names RDD:

1. map - to split the ID from the Name and store them in different columns
2. map - to put them in a tuple for easier access
3. filter - to get the Name of the hero with the ID with the most appearances in the dataset. **(Task 6)**
4. filter - to get the Name of the hero with the ID with the least appearances in the dataset. **(Task 7)**

Output:



```
The hero that appears the most is: "CAPTAIN AMERICA" with 1937 appearances!
```



```
The hero that appears the least is: "RED WOLF II" with 1 appearances!
```

Assignment 8

In this task, using Spark SQL, we apply the following operations:

Task 1: Find the minimum temperature per city

We start by creating a schema since the csv file doesn’t have one:

1. stationID (String)
2. date (Integer)
3. type (String)
4. temp (Float)
5. [Not used]:
   a. other (String)
   b. other1(String)
   c. other2 (String)

And then we apply a single SQL query:

```
SELECT stationID, MIN(temp)
FROM temps
WHERE type = 'TMIN'
GROUP BY stationID
```

Where we collect the stationID and perform the minimum function to collect the minimum temperature on the rows where the type value is TMIN, while isolating this operations in groups with different stationID. We end the task by printing the temperatures corrected by dividing them by 10.

Tasks 2 and 3: Obtain and sort the word frequency in a Book

We start by inserting the data in the Spark SQL and cleaning and splitting the text using spaces and store a word per row and strip the empty strings. Then, we perform the following query:

```
SELECT word, COUNT(*) as count
FROM words
GROUP BY word
ORDER BY count DESC
```

In which we count the occurrences of each word **(Task 2)** and then order the output by the count value in descending order, so the most popular word comes in top **(Task 3)**.

Tasks 4 and 5: Obtain and sort the total amount spent by customer

Starting by creating a schema for the data in the csv file that is missing a header, we have:

1. customerID (Integer)
2. productID (Integer)
3. amount (Float)

Then, we read the data and apply the schema. Proceeding to the query, we have:

```
SELECT
        customerID,
        SUM(amount) as sum
FROM purchases
GROUP BY customerID
ORDER BY SUM(amount) DESC LIMIT 10
```

where we output the customerID and the sum of the amount column for each customer **(Task 4)** and then sort it by the total amount of purchases **(Task 5)**.

Tasks 6 and 7: Find the most and least popular superhero in the dataset

We use two data frames to store the information from the two files, the *movies_df* and *names_df*. In the *movies_df* we store each ID appearance in a row, for an easier count operation. And in the *names_df* we store the ID and the hero's name in different columns.

Task 6 - To find the most popular superhero, we apply the following operations:

1. groupBy - to isolate the remaining operation by each hero
2. count - to count the number of occurrences
3. orderBy - to order the data frame in descending order of the count column value

Task 7 - To find the least popular superhero, we apply the same query, changing only the third step, in which we order by ascending order, leaving the least popular hero at the top of the data frame.

To conclude these two tasks, we perform one query for each task to fetch the name of the hero with the corresponding ID from the previous queries.

Assignment 9

Task 1: Obtain and sort the word frequency in the command line / message queue

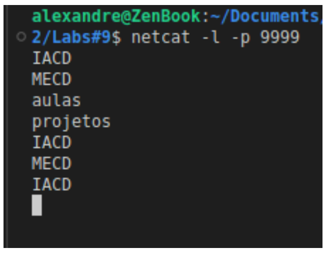
In this assignment, we deal with real-time processing by using Spark Streaming, where we create a socket in a terminal and then execute our script to perform the task.

We start by creating the socket and to get ready to send some words through the socket we run:

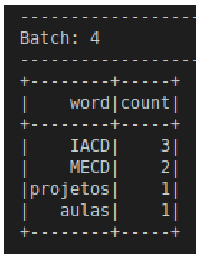
$ netcat -l -p 9999

Then we run our task which consists of a script, that loads the information in the socket and keeps adding it to the data frame since it is running in **complete mode**.
Since we are dealing with Spark Streaming, the task is performed every time the socket receives something.

Finally, the tasks that are performed, are the same as in Spark SQL, in which we perform the *explode* function to store a word per row and then the *groupBy* and *count* operation to get the word frequency of each word, proceeded by the *sorting* of the data frame in descending order. The only difference is the command used to display the *data frame,* in which we use the *writeStream* function in *complete mode* to show the counts after each batch arrival.

Input: 

```
alexandre@ZenBook:~/Documents/
2/Labs#9$ netcat -l -p 9999
IACD
MECD
aulas
projetos
IACD
MECD
IACD
```

Output: 

```
-------------------
Batch: 4
-------------------
+--------+-----+
|    word|count|
+--------+-----+
|    IACD|    3|
|    MECD|    2|
|projetos|    1|
|   aulas|    1|
+--------+-----+
```

Rui Alexandre Tapadinhas, 2018283200
Pedro Tavares, 2018280907