- Assignment 6

In this assignment, the major goal was to develop various tasks using Hadoop MapReduce.

Task #1 - Get the Average Number of Friends by Age

In this task, we were given a csv file that contained a list of people with their corresponding ages and the number of their friends.

Steps:

- Step Definition, being needed two steps in this resolution
- Mapper #1 had the goal to get the number of friends, splitting the file with commas “,” so that the information was comprehensible
- Reducer #1 was used to count the number of people that are that age and to make the average calculation with the total friends amount of everyone that is a certain age
- Reducer #2 was used to output the results into the command line

```
"62"    220.76923076923077
"63"    384.0
"64"    281.3333333333333
"46"    223.69230769230768
"47"    233.22222222222223
"68"    269.6
"69"    235.2
"65"    298.2
"66"    276.44444444444446
"67"    214.625
```

In order to save some space, I decided just to present a few of the results. In the image we can check the age followed by the average number of friends.

Task #2 - Get the Minimum Temperature per Station

For the second task of this assignment, we were given a csv file that contained some information about the weather registrations in various stations.

Steps:

- Step Definition, for this task I just needed a Mapper and a Reducer to be successful in obtaining the minimum temperatures per station
- Mapper - The mapper has the function to split the file and to extract the needed information for the task, here was the station, date, temperature type (whether it was minimum, maximum, between others), and the value of the temperature recorded. In this function, there is also a condition to check if the temperature type was minimum, saving only the values where the condition was met.
- Reducer - The reducer was used just to return the correspondent station and the minimum of the temperatures that passed the condition in the Mapper function.

```
"ITE00100554"   -148
"EZE00100082"   -135
```

In this image, there are two stations registered with their corresponding minimum temperatures.

Task #3 - Sort the Word Frequency in a Book

In this third task, there were a few more steps required because a book has a lot of punctuation and one simple character can cause the count to go up and make it wrong.

Steps:

- Step Definition, since there were more tasks to do, there was the need to add two mappers and two reducers.
- Mapper #1 - The first mapper has the function to get all the lines from the document and to strip all the punctuation before splitting them up so that they become words. And, to conclude its job the mapper function minimizes all the capital letters present in the words.
- Reducer #1 - The first reducer’s job is to count all the times a word appears
- Mapper #2 - Sorts all the words and its count
- Reducer #2 - Outputs the sorted words one by one

```
"gutenberg"     3
"alone" 3
"horses"        3
"huge"  3
"indeed"        3
"strangers"     4
"take"  4
"than"  4
"just"  4
"little"        4
```

The image shows some words present in the book, in order to optimize space I have limited the count to only 10 so that no space is wasted.

Task #4 - Sort the Total Amount Spent by Customer

Just like the tasks above, in this task we were given a csv file that contained information about customers and their orders.

Steps:

- Step Definition, for this task, there was the need to get two Mappers and two Reducers in order to get all the correct data and organized
- Mapper #1 - Just like in the other tasks, this mapper function is needed to split all the relevant information with a comma “,” so that it can be easily accessed.
- Reducer #1 - Gets the amount spent by customer
- Mapper #2 - This mapper receives None as the key, once the sorting is going to be generalized and through the whole dataset
- Reducer #2 - Prints every pair of key, values with the sorted amount spent and then returns it.

```
"68"    6375.449999999997
"73"    6206.199999999999
"39"    6193.109999999999
"54"    6065.389999999999
"71"    5995.660000000003
"2"     5994.59
"97"    5977.189999999995
"46"    5963.109999999999
"42"    5696.840000000003
"59"    5642.89
```

In this picture, I selected the top 10 customers with the most quantity spent on orders to show the supposed output of this task, trying to not waste valuable space.

- Assignment 7 and 8

For this two Assignments the tasks to be developed were the same but with different techniques. Being the Assignment 7 made to use Spark RDD and Assignment 8 to used Spark SQL.

Task #1 - Get the Minimum Temperature per Station

Spark RDD Approach

- Create the SparkConf and SparkContext, read the file and choose its path
- Analyse Entries: In this step, a definition called parseLine received the lines from the file and separated them with commas, so that all the fields were separated such as ID was fields[0], entryType was fields[2], and temperature was a float(fields[3]). This indicates the position of those attributes in the line.
- Filter Information: After the data separation was complete, it was time to apply a filter in order to get only the TMIN entries, because we only want to get the minimum temperature.
- Mapping: After the filter was concluded, we only needed the station ID and the value of the temperature, making only a map of those two columns.
- Reduce by key: After the mapping phase, I reduced those entries by key, making them only have the value of the minimum temperature registered and not all the TMIN entries.

```
('ITE00100554', -148.0)
('EZE00100082', -135.0)
```

In this image, the results of this task are presented.

Spark SQL Approach

- Create a Spark Session, and file configuration
- Create a personalized schema for this dataset using StructType and StructFields
- Reading the csv file with the applied schema and selecting only the important information
- Just like in the RDD approach, I filtered all the non TMIN entries in the entryType column out so that the dataset is all cleaned
- Query: Select the stationID and temperature, Group by StationID and only show the minimum value of temperature

```
+-----------+----------------+
|  stationID|min(temperature)|
+-----------+----------------+
|ITE00100554|          -148.0|
|EZE00100082|          -135.0|
+-----------+----------------+
```

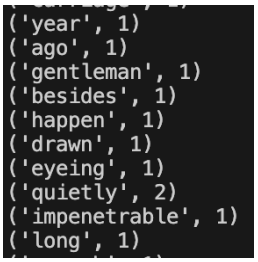
Task #2 and #3 - Get the word frequency of a book and then sorted

The construction of these two tasks were basically the same since the only change was that after getting the frequency there was the need to organize them and sort them.

Task #2

Spark RDD Approach

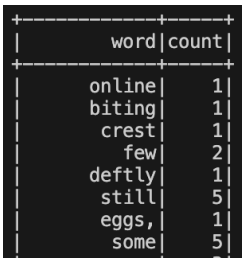
- Get all the configurations done
- Analyse the Book: In this step, the file that contained the book had a few transformations done to it. Being any upper case lowered and then the lines were split
- To get the frequency of the words, there was the need to apply a flatMap to the analysed file and then to reduce them by key summing all the times the word came up.
- Before printing the results, there was an action to collect them.



In this image there is the output of the steps explained above.

Spark SQL Approach

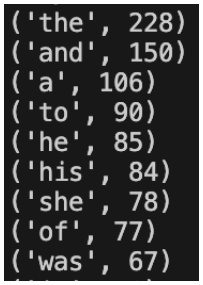
- For this approach, I defined the file path and then read the lines using spark with the header option with a false value because the book wasn't a csv file with column names, for example
- For the processing of this file, the pyspark functions were very helpful. The explode function that made the whole text be separated into lines, the split that made that every word was alone and the lower that made every capital letter go low. Using alias to give the name word to the column that was created with all the words.
- Query: Create a temporary table named word table where the words, and the count of the occurrences were selected from and then grouped by word.



Output of the query and steps explained above.

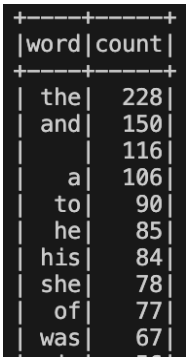
Task#3

Spark RDD Approach



The only change that was made from the past task to this one was the sort by made after the reduce by key in order to get all the values sorted descending. Having the output shown on the image in the left.

Spark SQL Approach



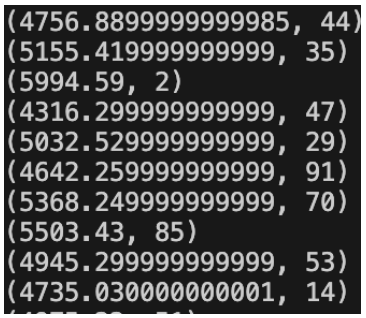
word	count
the	228
and	150
	116
a	106
to	90
he	85
his	84
she	78
of	77
was	67

As the last task, in this one, there wasn't a lot to do, the only thing needed to order the values in descending order was to add a line to the query. This line makes all the difference, having a more clean work and perceptible for everyone.

Tasks#4 and #5
In these pair of tasks, the main goal was to get the total amount spent by customer, having task #4 not sorted and then task #5 sorted out.

Task #4
Spark RDD Approach
Steps:

- Configuration of spark
- Lines Analysis: Select the important fields, and their types and split the data with commas ","
- Map: Makes the application of the functions above defined
- Reduce by Key: Combines rows with the same key and applies the function defined, in this case, the sum
- Collect Action: Collects the results from the Transformations operations above defined

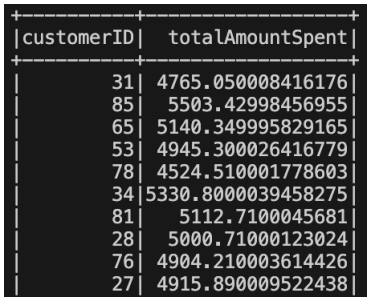


```
(4756.8899999999985, 44)
(5155.419999999999, 35)
(5994.59, 2)
(4316.299999999999, 47)
(5032.529999999999, 29)
(4642.259999999999, 91)
(5368.249999999999, 70)
(5503.43, 85)
(4945.299999999999, 53)
(4735.030000000001, 14)
```

This image illustrates the output that the steps above create.

Spark SQL Approach
Steps:

- Personalized Schema: I decided to construct a personalized schema for this dataset because it was easier to make all the other operations and applied it to the file.
- Temporary Orders table creation that represents the whole dataset so that the query can be applied
- Query: Select customerID, the sum of the spent amount from orders and group those values by customer ID resulting in the image below.



customerID	totalAmountSpent
31	4765.050008416176
85	5503.42998456955
65	5140.349995829165
53	4945.300026416779
78	4524.510001778603
34	5330.8000039458275
81	5112.7100045681
28	5000.71000123024
76	4904.210003614426
27	4915.890009522438

Task #5
Spark RDD Approach

```
(6375.449999999997, 68)
(6206.199999999999, 73)
(6193.109999999999, 39)
(6065.389999999999, 54)
(5995.660000000003, 71)
(5994.59, 2)
(5977.189999999995, 97)
(5963.109999999999, 46)
(5696.840000000003, 42)
(5642.89, 59)
```

In this approach for this problem, the only thing that changed from the above was in the reduce by key, the definition of ascending equals false meaning the order was going to be descending.

Spark SQL Approach

```
+----------+------------------+
|customerID|  totalAmountSpent|
+----------+------------------+
|        68| 6375.450028181076|
|        73| 6206.199985742569|
|        39| 6193.109993815422|
|        54| 6065.390002984554|
|        71| 5995.659991919994|
|         2| 5994.589979887009|
|        97| 5977.190007060766|
|        46| 5963.110011339188|
|        42| 5696.840004444122|
|        59| 5642.890004396439|
```

In the SQL approach, the only thing that changed to get the output shown was the addition of a line in the query that ordered the total amount spend descending.

Task #6 and #7
The main goal of these tasks were to present the most popular superhero and the least popular one. Being the basis of the code the same, the steps only change when trying to identify which one is which.
Spark RDD Approach
Steps:

- Creation of two RDDs one for the Hero count and the other for the Names since they come from different files.
- In the Hero rdd, several operations are needed such as a flatMap to separate the numbers from each line, a filter to remove empty entries, a map to transform each pair of hero and counter. The counter (that begins in 1 to start the count correctly) and a reduceByKey to sum the counters. A sortBy to sort the occurrence in descending order in order to the most popular heros to appear on the top, or ascending order if were on Task 7 for the least popular hero to appear.
- For the other RDD, which has the names of the heroes it was needed a map to store the id and the name in different columns. Another map for better and easier access and two filters one for each task. One for the Id with the most appearances and one for the ID with least appearances

Output:
Task 6

```
Most Popular: "CAPTAIN AMERICA"
```

Task 7

```
Least Popular: "RED WOLF II"
```

Spark SQL Approach
Steps:

- In this approach first, there is the reading of one file and its splitting and explosion so that the names for the heroes can be all by themselves, one in each line without the ID. While applying that, there is also a filter going that filters whether the column which has the names has empty values or not and only consider the ones where there is content.
- For the next file, first there is the reading and splitting, considering two words after the split. After this, there is the collection of information from the dataset so that we can apply the query later
- Before the application of the query, I decided to join the two datasets where they meet, which is the hero ID that is present in both variables.
- Query for the most popular superhero: Select the name and count of times he appears, just to be sure that we are not choosing some random hero that has almost never appeared on the screen.
- For the output, it is limited to one because we are just looking for the most popular, and the order by is descending so that the first one is definitely the most popular.
- The query for the least popular superhero is exactly the same, only changes the order by, that becomes ascending so that the first one is without any doubt one of the least popular heroes.

In the two tasks 7, the result is different because there are a few more that only have appeared one time on the movies.
Output:
Task 6

```
+------------------+-----+
|          heroName|count|
+------------------+-----+
|"CAPTAIN AMERICA"| 1937|
+------------------+-----+
```

Task 7

```
+---------------+-----+
|       heroName|count|
+---------------+-----+
|"BERSERKER II"|    1|
+---------------+-----+
```

- Assignment 9

For this assignment, the main goal was to have a Spark Streaming obtaining and sorting the word frequency from the command line.
Steps:

- Set up a streaming source with the help of netcat in a socket format listening for incoming data on a specific host and port (localhost, 9999)
- Split the words if the user puts two words in the same line separated by a whitespace
- Count the words, Order by count
- Query to write the stream of words

Results:

```
+-----+-----+
| word|count|
+-----+-----+
| iacd|    4|
|aulas|    2|
|verde|    1|
|teste|    1|
+-----+-----+
```

```
marianapaulino@Marianas-MBP-4 La
bs#9 % nc -l -p 9999
aulas
iacd
teste
aulas
iacd
verde
iacd
iacd
```