Exercise 1 : Spark Fundamentals

Question 1

Discuss in group the concept of laziness in Spark. Why is it useful ?
Come up with an example in which laziness is greatly beneficial.

Answer

Laziness is useful to queue up computations to:

1. *reduce network calls, and*
2. *perform optimizations such as pipelining on staged computations.*

Question 2

Assume that you have the following `RDD` of strings:

```
val rdd: RDD[String] = ...
```

Efficiently compute the number of words in your RDD. In your solution, highlight the transformations and actions.

Hint: You may use the method `.split(separator: String)` on strings to split a string based on a `separator`. You may assume that words are separated by spaces only.

Answer

```
rdd.map(text => text.split(" ").length).reduce(_ + _)
```

Question 3

Assume that you are given the following RDD of strings that holds raw logs (info, warning and error messages).

```
val rawLogs: RDD[String] = ...
```

In addition, assume that you are given the following methods:

```
def toLog(str: String): LogEntry = ...
def isError(log: LogEntry): Boolean = ...
def message(log: LogEntry): String = ...
```

Your goal is to *efficiently* compute the following values:

1. The number of errors in the logs.
2. An array containing the message of all errors.

Answer

```
val errorLogs = rawLogs.map(toLog(_)).filter(isError(_)).cache()
val numberErrors = errorLogs.count()
val messages = errorLogs.map(message(_)).collect()
```

Question 4

Assume that you are given an `RDD` of strings called `rdd`. Foreach value, print it to the standard output of the node that holds this value.

Answer

```
rdd.foreach(println(_))
```

Exercise 2 : Demographics

Imagine that you work in a large software company, whose main product is a social media website. In this exercise, your goal will be to compute demographical data from the dataset of all millions users of the website. You may assume that users entries are of form `Person`. This data type is defined as:

```
case class Person(age: Int, name: String)
```

The `RDD` that holds all the entries is called `people`.

Question 1

Your goal in this first part is to first exclude of the dataset all people that are under 18, and only keep the age. *(It would certainly not be ethical to work on children, wouldn't it?)*

Then, you will have to count how many users fall in the following age groups:

```
1: 18-25
2: 26-35
3: 36-45
4: 46-65
5: 66+
```

You may safely assume that no user in your website is older than 200 years.

For this question, we ask you to iterate through the filtered dataset 5 times – once for each group – and count the number of people in the group. Your goal is to end up with a list of the counts. You may assume that you are given the following list:

```
val groups = List((18, 25), (26, 35), (36, 45), (46, 65), (66, 200))
```

Answer

```
val adultAges = people.map(_.age).filter(_ >= 18).cache()
groups.map {
  case (lower, upper) => groups.filter { (age: Int) =>
    lower <= age && age <= upper
  }.count()
}
```

Question 2

Come up with a solution that does not need to iterate the dataset 5 times. You may make use of the following function in your solution:

```
def groupOf(age: Int): Int = ...
```

This function will return the number of the group associated to the age. For instance, it will return 2 for the age 27, and 4 for 46.

As before, your goal will be to return a list of all counts. Which of the two versions do you think would be more efficient ? Why ?

Answer

```
people.map(_.age)
      .filter(_ >= 18)
      .map((age: Int) => (groupOf(age), 1))
      .reduceByKey(_ + _)
      .collect()
      .toList()
      .sortBy(_._1)
      .map(_._2)
```