Big Data Computing

Homework 1

You will develop a Spark program to analyze a dataset of an online retailer which contains several transactions made by customers, where a transaction represents several products purchased by a customer. Your program must be designed for very large datasets.

DATA FORMAT

The dataset is provided as a file where each row is associated to a product purchased in a transaction. More specifically, a row consists of 8 comma-separated fields: *TransactionID* (a string uniquely representing a transaction), *ProductID* (a string uniquely representing a product), *Description* (a string describing the product), *Quantity* (an integer representing the units of product purchased), *InvoiceDate* (the date of the transaction), *UnitPrice* (a real representing the price of a unit of product), *CustomerID* (an integer uniquely representing a customer), *Country* (a string representing the country of the customer). For example the row: **A536370,21883B,STAR GIFT TAPE, 24,2010-12-01 8:45,0.65,12583,France** represents the fact that Transaction A536370 made by Customer 12583 on 12/1/2010 at 8.45am contains 24 units of Product 21883B called Star Gift Tape. If in the same transaction the customer purchased several products, they will be represented in separate rows. Note that the Quantity field can also contain a negative value $-x$ to denote that x previously purchased units of the product have been returned.

TASK

You must **write a Java program** (*hw1.java*) which receives in input, as command-line (CLI) arguments, 2 integers **K** and **H**, a string **S**, and a path to the file storing the dataset, and does the following:

1. Reads the input file into an **RDD of strings** called **rawData** (each 8-field row is read as a single string), and subdivides it into **K partitions**, and prints the number of rows read from the input file (i.e., the number of elements of the RDD).

2. Transforms rawData into an **RDD of (String,Integer) pairs** called **productCustomer**, which contains all *distinct pairs* (P,C) such that rawData contains one or more strings whose constituent fields satisfy the following conditions : ProductID=P and CustomerID=C, Quantity>0, and Country=**S**. If S="all", no condition on Country is applied. It then prints the number of pairs in the RDD. **IMPORTANT**: *since the dataset can be potentially very large, the rows relative to a given product P might be too many and you must not gather them together; however, you can safely assume that the rows relative to a given product P and a given customer C are not many (say constant number). Also, although the RDD interface offers a method distinct() to remove duplicates, we ask you to avoid using this method for this step.*

3. Uses the mapPartitionsToPair/mapPartitions method (combined with the groupByKey and mapValues or mapToPair/map methods) to transform productCustomer into an **RDD of (String,Integer) pairs** called **productPopularity1** which, for each product ProductID contains one pair (ProductID, Popularity), where Popularity is the number of distinct customers from Country S (or from all countries if S="all") that purchased a positive quantity of product ProductID. **IMPORTANT**: *in this case it is safe to assume that the amount of data in a partition is small enough to be gathered together.*

4. Repeats the operation of the previous point using a combination of map/mapToPair and reduceByKey methods (instead of mapPartitionsToPair/mapPartitions) and calling the resulting RDD **productPopularity2**.

5. (This step is executed only if H>0) Saves in a list and prints the ProductID and Popularity of the **H products with highest Popularity**. Extracts these data from **productPopularity1**. Since the RDD can be potentially very large *you must not spill the entire RDD onto a list and then extract the top-H products.*

6. (This step, for *debug purposes*, is executed only if H=0) Collects all pairs of **productPopularity1** into a list and print all of them, in increasing lexicographic order of ProductID. Repeats the same thing using **productPopularity2**.

To test your program you can use the 3 provided datasets, in the same section as this specification, together with the outputs of some runs of the program on the datasets.