Working with MapReduce

DS730

In this project, you will be working with input, output, Python and Hadoop framework. You will be writing multiple mappers and reducers to solve a few different problems. If you recall, the map and reduce functions are **stateless** and this is especially important when dealing with Hadoop and distributed work. We can't guarantee that any 1 mapper will read in all of the data. Nor can we guarantee that certain inputs will end up on the same machine for mapping. Rather, 1 mapper will likely read in a small portion of the data. The output that your mapper produces must only depend on the current input value. For the reducer, you can only guarantee that (key,value) pairs with the same key will end up on the same reducer.

Your mapper and reducer cannot be trivial. For example, do not have all of your mappers map use the same key and then solve everything in the reducer. Such a solution defeats the purpose of MapReduce because all (key,value) pairs will end up on the same reducer. If you are unsure if your keys are trivial, post a private message to the message board for the instructors and we will let you know if your keys are trivial. A couple of very important things:

1. **Make sure your key is separated by your value using a tab. Hadoop will only work if this is the case. Otherwise, Hadoop has no idea what your "key" is nor will it know what your "value" is.**
2. **Make sure that this is the first line of your mapper and reducer: #!/usr/bin/python3.6**

You must write 1 mapper file and 1 reducer file to solve each of the following problems. Make sure you name your files **mapperX.py** and **reducerX.py** where X is the problem number. For each problem, Hadoop will define what your input files are so there is no need to read in from any file. Simply read in from the command line. You are encouraged to use the "starter" mapper and reducer as shown in the activity.

Problem 1: Finding Big Spenders (50 points)

Assume you work for a large business and have access to all orders made in any given time period (download the orders.csv file from

http://faculty.cs.uwosh.edu/faculty/krohn/ds730/orders.csv)[1]. When we test your code, the input file will have an identical format to the orders.csv file[2]. The column headings are fairly self-explanatory. Your company wants you to find the *big spenders* for each month and country so they can market more heavily to them. Your goal is this: for each month/country combination, display the customerID of the top spender (i.e. the sum of how much that customer spent) for that month/country combination. The amount spent in each row is determined by multiplying the Quantity by the UnitPrice. A few caveats:

a. The InvoiceDate is in Month/Day/Year format.
b. An InvoiceNo that starts with a C is a return. You must ignore these rows.
c. A row may not have a CustomerID. These rows must be ignored.
d. We are **not** looking for month/year/country combinations here. We are only interested in the month/country combination.

This final output file should contain the following data in this format:

Month,Country:CustomerID

The month must be a two-digit number (i.e., 01, 02, ..., 09, 10, 11, 12) and must be separated from the Country using a comma. The Month,Country portion is separated by the CustomerID using a colon. If there is a tie, you must print out all customers who tied separating the CustomerID's by a comma with the CustomerId's being in ascending order.

Problem 2: Suggesting Products (70 points)

We will be using a modified orders.csv file from Problem 1. You can download the orders.csv file for this file here: http://faculty.cs.uwosh.edu/faculty/krohn/ds730/ordersProblem2.txt. In this file, you will find that each line contains the following:

InvoiceNumber1,InvoiceNumber2…!StockCode1:StockCode2:StockCode3…

The StockCode's are the items bought for a particular InvoiceNumber. If an order was duplicated, then the InvoiceNumber of all duplicated orders appear at the start of the line and are separated by commas. For example, let's say InvoiceNumber 1 ordered

[1] Data set used: Daqing Chen, Sai Liang Sain, and Kun Guo, Data mining for the online retail industry: A case study of RFM model-based customer segmentation using data mining, Journal of Database Marketing and Customer Strategy Management, Vol. 19, No. 3, pp. 197-208, 2012 (Published online before print: 27 August 2012. doi: 10.1057/dbm.2012.17)
[2] In other words, there will be a header row and there will be any number of rows of actual data.

items 100, 101 and 103. Let's assume InvoiceNumber 8 ordered items 100, 101 and 103. In the input file, this would be denoted as:

1,8!100:101:103

Instead, if InvoiceNumber 8 would have ordered items 100, 101 and 106, the the input would look like this:

1!100:101:103
8!100:101:106

There will be at least 1 InvoiceNumber at the start of each line with an unbounded number of InvoiceNumbers potentially appearing after that. Once the InvoiceNumbers for all duplicated orders are finished, an exclamation point separates the InvoiceNumbers from the StockCodes. A StockCode is a number associated with an item bought. The StockCodes are separated by colons. At least 1 StockCode will exist with an unbounded number of StockCodes potentially appearing after that.

In this problem, we want to come up with suggestions of products for people to buy. The goal of this problem is to output the following:

StockCode - StockCodeSuggestion1,StockCodeSuggestion2,…

If two items were bought together on at least 5 different invoices, then we are going to assume that those two items should be suggested for each other. For example, consider the following input:

1,2,3,4!100:101:102:103
5,6!101:103:106:107
7,8,9!102:106:107:110
10,11,12!101:105:108:109
13!106:110
14!100:102:103:110

In the previous example, 101 and 103 were bought together on 6 different orders. Namely, order numbers 1, 2, 3, 4, 5 and 6. Therefore, in the output, 101 should be a suggestion for 103 and 103 should be a suggestion for 101. However, 106 and 110 were only bought in orders 7, 8, 9 and 13. Therefore, they should not be suggestions for each other because they did not meet that "at least 5 different invoices" threshold. The output of the previous example would be:

100 : 102,103
101 : 103

102 : 100,103
103 : 100,101,102
106 : 107
107 : 106

The output must be in the following format:

StockCode : StockCodeSuggestion1,StockCodeSuggestion2,…

In other words, the StockCode, followed by a space, followed by a colon, followed by a space, followed by a comma separated list of items that the StockCode should be suggested with. The StockCodes to the right of the dash must be in numerical order from lowest to highest. If the number has a letter at the end, then the number without the letter comes before the one with the letter. For example, 1234 would come before 1234A and 1234A would come before 1234B. Finally, 1234B would come before 1235. If a StockCode has nothing that it pairs with, then it should not appear in the output. For example, in the above example, 108 has no suggestions. Therefore,

108 :

should not appear in the output.

Other Important Information

1. You can use any Python libraries as long as they are installed by default on the Hortonworks machine and your code works on Hadoop. **All projects will be tested using Hadoop on Hortonworks. You should ensure that your code executes correctly on that platform before submitting.**
2. Although not a coding question per se, asking publicly about (key,value) pairs or what keys one should use is a major question that should be done privately. If you know what the (key,value) pair is, then it's really just a programming task of implementing MapReduce correctly. Coming up with the (key,value) pair is a crux of solving these problems. Therefore, if you have a question about whether or not your key is good, please make it a private question.
3. When you are working with Hadoop and other software at your job, you will likely be given some large dataset and have to work with it. This is the case with number 1. However, you may not be given a set of sample input and output files to test your code. Therefore, one of the objectives for this project is being able to test your solution without having a sample input/output file. Coming up with your

own sample input and output based on the input description (and the specified output) is a key skill to have. These samples can be completely random or you can generate them using some real world data. You can expand some of the examples given in the questions and manually check if your answer is correct. You can also make smaller input files for problem 1 and then manually check if your answer is correct. If it is correct for several small cases, then you should feel good about your answer being correct for a large case that you can't manually check.

4. The order of the rows in the output does not matter. This is shown in the last part of problem 2.
5. Your code must run correctly on Hadoop. If your code is not stateless or crashes on Hadoop, you will earn at most 50% for this project. Getting your code to run using this framework is one of the main goals of this project. If it doesn't at least do that, there isn't much credit to be earned for this project.
6. Your code must, at minimum, run correctly using regular Python. If your code doesn't even work with regular Python, you will earn at most 25% for this project. Getting your code to run with regular Python (outside of the Hadoop framework) should be the bare minimum.

What to Submit

When you are finished testing your code, zip up your mappers and reducers for the problems into a zipped file called `p1.zip` and upload only this zip file to the Project 1 dropbox. As a reminder, make sure you name your files **mapperX.py** and **reducerX.py** where X is the problem number.