Working with Hadoop

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/env python**

You must write a mapper file and a 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 (15 points)

(20 pts) 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://www.uwosh.edu/faculty_staff/krohne/ds730/orders.csv)[1]. We will use this exact csv file when testing your code. 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. I am **not** looking for month/year/country combinations here. I am only interested in the month and country.

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.

Problem 2: Words with exact same vowels (15 points)

This problem is similar to the problem we worked on in lecture with a small twist. Instead of printing out how many times a word appears in the file, you want to print out how many words have the exact same type of vowels. For this problem, only the number of vowels matters and the case does not matter (i.e **cat** is the same as **CAt**). A vowel is any letter from this set {a,e,i,o,u,y}. **A *word* is any sequence of characters that does not contain whitespace**. Whitespace is defined as: space, newline or tab. All of the following are 1 single word:

cats
c@ts
ca7s
cat’s.and:d0gs!

[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. 197â€“208, 2012 (Published online before print: 27 August 2012. doi: 10.1057/dbm.2012.17)

The output will be the vowel set, followed by a colon, followed by the number of words that contained exactly the vowel set. The output will have one answer per line (see example below).

Example:

‘hello’ and ‘pole’ both contain exactly 1 e and exactly 1 o. The order of the vowels from the original input word does not matter.

Imagine the following example:

```
hello    moose
pole cccttt.ggg
```

We would end up with the following output:

```
:1

eo:2

eoo:1
```

The format should be as seen above: the vowels on each line are in alphabetical order, followed by a colon, then followed by the number of words that contained exactly those vowels. If there are words with no vowels, nothing is printed before the colon.

Problem 3: Discovering Contacts (20 points)

On many social media websites, it is common for the company to provide a list of suggested contacts for you to connect with. Many of these suggestions come from your own list of current contacts. The basic idea behind this concept being: I am connected with person A and person B but not person C. Person A and person B are both connected to person C. None of my contacts are connected to person D. It is more likely that I know person C than some other random person D who is connected to no one I know. For this problem, all connections are mutual (i.e. if A is connected to B, then B is connected to A). In this problem, you will read in an input file that is delimited in the following manner:

PersonA : PersonX PersonY PersonZ PersonQ
PersonB : PersonF PersonY PersonX PersonG PersonM
…

For example, the person to the left of the colon will be the current person. All people to the right of the colon are the people that the current person is connected to. All people will be separated by a single space. In the example above, PersonA is connected to

PersonX, Y, Z and Q. In all inputs, all people will be replaced with positive integer ids to keep things simple. The following is a sample input file:

```
1 : 3 5 8 9 10 12
2 : 3 4 7 6 13
3 : 9 11 10 1 2 13
4 : 2 5 7 8 9
5 : 4 1 7 11 12
6 : 2 9 8 10
7 : 5 2 4 9 12
8 : 1 6 4 11
9 : 12 1 3 6 4 7
10 : 1 3 6 11
11 : 3 5 10 8
12 : 1 7 5 9
13 : 2 3
```

The ordering of people on the right hand side of the input can be in any order. Your goal is this: you must output *potential contacts* based on the following 2 criteria:

i. Someone who **might** be someone you know. For someone to be suggested here, the person must not currently be a connection of yours and that person must be a connection of exactly 2 or 3 of your current connections. For example, consider person 2 in the above example. Person 2 is connected with 3, 4, 6, 7 and 13. Person 4 is connected to 8, person 6 is connected to 8, person 3 is not connected to 8, person 7 is not connected to 8 and person 13 is not connected to 8. Therefore, person 2 has two connections (4 and 6) that are connected to 8 and person 2 is not currently connected to 8. Therefore, person 2 might know person 8.

ii. Someone you **probably** know. For someone to be suggested here, the person must not currently be a connection of yours and that person must be a connection of 4 or more of your current connections. For example, consider person 2 in the above example. Person 2 is connected with 3, 4, 6, 7 and 13. Person 4 is connected to 9, person 6 is connected to 9, person 3 is connected to 9 and person 7 is connected to 9. Therefore, person 2 has at least four connections that are connected to 9 and person 2 is not currently connected to 9. Therefore, person 2 probably knows person 9.

Your output must be formatted in the following fashion:

```
personID:Might(personA,…,personX) Probably(personA,…personX)
```

For each line you have a personID following by a colon. The colon is followed by the list of Might’s separated by commas (but no space). If a person has no one they might be connected to, this list is not printed at all (see person 13 below for example). The Might list is followed by a single space and then followed by the Probably list separated by commas (but no space). If a person has no one they probably are connected to, this list is not printed at all (see person 3 for example). If a person has neither a might list nor a probably list, that person only has their id along with a colon (see person 13 for example). The *Might* list must appear before the *Probably* list. If there is no *Might* list but there is a *Probably* list, there is no space between the colon and the *Probably* list. The integers within each list must appear in increasing order. However, the order the rows appear in the output need not be in any specific order. For example, the row for 5 might appear before the row for 3. As a concrete example from the above sample input, this would be a potential sample output:

```
1:Might(4,6,7) Probably(11)
2:Might(5,8,10) Probably(9)
3:Might(4,5,6,7,8,12)
4:Might(1,3,6,11,12)
5:Might(2,3,8,10) Probably(9)
6:Might(1,3,4,7,11)
7:Might(1,3,6)
8:Might(2,3,5,9,10)
9:Might(8,10) Probably(2,5)
10:Might(2,5,8,9)
11:Might(4,6) Probably(1)
12:Might(3,4)
13:
```

The rows do not have to be in any specific order. The following is also a valid output:

```
9:Might(8,10) Probably(2,5)
1:Might(4,6,7) Probably(11)
3:Might(4,5,6,7,8,12)
4:Might(1,3,6,11,12)
```

```
5:Might(2,3,8,10) Probably(9)
11:Might(4,6) Probably(1)
6:Might(1,3,4,7,11)
7:Might(1,3,6)
2:Might(5,8,10) Probably(9)
8:Might(2,3,5,9,10)
10:Might(2,5,8,9)
12:Might(3,4)
13:
```

Other Important Information

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.

All projects will be tested using Hortonworks on Azure. You should ensure that your code executes correctly on that platform before submitting.

What to Submit

When you are finished testing your code, zip up your mappers and reducers for all problems into a zipped file called `p1.zip` and upload only this zip file to the Project 1 dropbox.