DATASCI W261: Machine Learning at Scale

Nick Hamlin and Tigi Thomas
nickhamlin@gmail.com, tgthomas@berkeley.edu
Time of Submission: 9:23 PM EST, Wednesday, Feb 3, 2016
W261-3, Spring 2016
Week 3 Homework

Submission Notes:

- For each problem, we've included a summary of the question as posed in the instructions. In many cases, we have not included the full text to keep the final submission as uncluttered as possible. For reference, we've included a link to the original instructions in the "Useful Reference" below.
- Problem statements are listed in *italics*, while our responses are shown in plain text.
- We've included the full output of the hadoop jobs in our responses so that counter results are shown. However, these don't always render nicely into PDF form. In these situations, please reference the complete rendered notebook on Github (https://github.com/nickhamlin/mids_261_homework/blob/master/HW3/MIDS-W261-2015-HWK-Week03-Hamlin-Thomas.ipynb)

Useful References:

- **Original Assignment Instructions (https://www.dropbox.com/sh/uev2esasn7bvtnd/AAAlxSC1J3ZHCm7EgkJbemAZa/H Questions.txt?dl=0)**
- Counter Example in mrjob (http://nbviewer.jupyter.org/urls/dl.dropbox.com/s/5thl14n4pqvhzt5/Counter.ipynb)

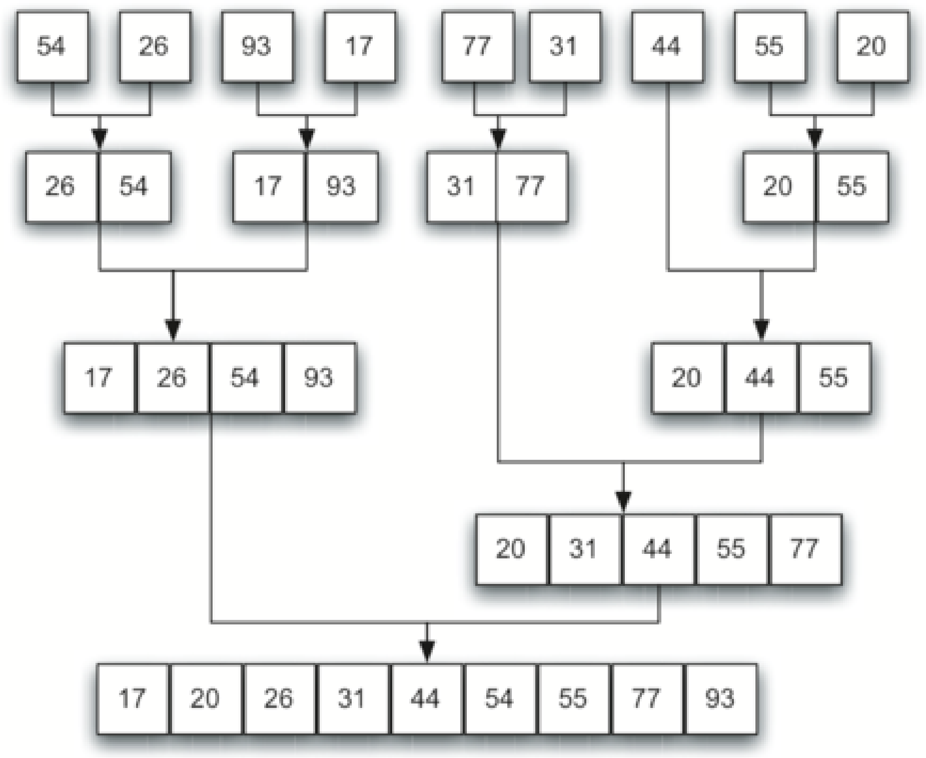
Handy Hadoop Links:

- Jobtracker (http://localhost:8088/cluster)
- Namenode (http://localhost:50070/dfshealth.html#tab-overview)

HW3.0.

What is a merge sort? Where is it used in Hadoop?

Merge sort is an algorithm that takes two sorted lists and merges them into a single sorted list. It does this by tracking two pointers at the top of each list. In each iteration, the values of the two pointers are compared, and the smaller of the two is appended to the final list. Therefore, as the algorithm iterates, the end result is a single sorted list. If the input to merge sort is an unsorted list, it must first be broken up into single elements, which essentially are (very small) sorted lists that can then proceed through the algorithm. This schematic from interactivepython.org (http://interactivepython.org/runestone/static/pythonds/SortSearch/TheMergeSort.html) helps make the process clear.



Hadoop uses merge sort during the shuffle process to take the multiple spill files that are generated for each partion and merge them together before sending them to the combiner. Merge sort is also used on the reducer side to combine files received from multiple mappers into a single stream.

How is a combiner function in the context of Hadoop? Give an example where it can be used and justify why it should be used in the context of this problem.

A combiner function is used to combine records with the same key before they are transferred to the reducer. For example, in a word count implementation, if a mapper generates two key-value pairs for the same word, a combiner would combine those two records together into a single pair with the same key and a value representing the sum of the two input values. In Hadoop, combiners can be used to reduce the amount of information that must be sent via the network from the mappers to the reducers. In the previous word count example, if combiners were omitted, the mappers would have to send two pairs across the network. But, by adding a combiner, only one pair per word needs to be sent.

This reduction in traffic can make a dramatic difference in the processing speed associated with a particular job, especially at large scale. In addition, if a combiner is available, Hadoop may or may not decide to use it in the execution of a particular job, so mission-critical logic should not be included in them and should be saved for the mapper or reducer.

What is the Hadoop shuffle?

The shuffle is the process by which Hadoop sorts the output of the mappers and transfers it to the reducers. It consists of three main steps: the partition, the sort, and the combine. The partition step takes the output of the mapper and partitions the results, by key, into separate files (one file for each reducer). In the sort step, records within the same partition are sorted. Finally, the combine step takes records with the same key and merges them together. Once these steps are completed, the output can then be transferred to the reducer.

HW3.1. Using Counters to do EDA

Counters are lightweight objects in Hadoop that allow you to keep track of system progress in both the map and reduce stages of processing. By default, Hadoop defines a number of standard counters in "groups"; these show up in the jobtracker webapp, giving you information such as "Map input records", "Map output records", etc.

While processing information/data using MapReduce job, it is a challenge to monitor the progress of parallel threads running across nodes of distributed clusters. Moreover, it is also complicated to distinguish between the data that has been processed and the data which is yet to be processed. The MapReduce Framework offers a provision of user-defined Counters, which can be effectively utilized to monitor the progress of data across nodes of distributed clusters.

*The consumer complaints (https://www.dropbox.com/s/vbalm3yva2rr86m/Consumer_Complaints.csv?dl=0) dataset consists of diverse consumer complaints, which have been reported across the United States regarding various types of loans.

User-defined Counters

Now, let’s use Hadoop Counters to identify the number of complaints pertaining to debt collection, mortgage and other categories (all other categories get lumped into this one) in the consumer complaints dataset. Basically produce the distribution of the Product column in this dataset using counters (limited to 3 counters here).

Hadoop offers Job Tracker, an UI tool to determine the status and statistics of all jobs. Using the job tracker UI, developers can view the Counters that have been created. Screenshot your job tracker UI as your job completes and include it here. Make sure that your user defined counters are visible.

HW 3.1 - Mapper and Reducer

Here, we simply use the type of product that the mapper sees to iterate the appropriate counter. After that, the reducer just passes the results through, since we're primarily interested in the counter outputs, not the data itself

In [247]:
```
%%writefile mapper.py
#!/usr/bin/python

#HW 3.1 - Mapper Function Code
import sys
for line in sys.stdin:
    line=line.strip()
    product=line.split(',')[1] #extract product field from second field

    #Iterate the counter depending on the product
    if product=='Debt collection':
        sys.stderr.write("reporter:counter:Debt,Total,1\n")
    if product=='Mortgage':
        sys.stderr.write("reporter:counter:Mortgage,Total,1\n")
    else:
        sys.stderr.write("reporter:counter:Other,Total,1\n")
    print product+'\t1'
```

```
Overwriting mapper.py
```

In [248]:
```
%%writefile reducer.py
#!/usr/bin/python

#HW 3.1 - Reducer Function Code
import sys
for line in sys.stdin:
    line=line.strip()
    print line
```

```
Overwriting reducer.py
```

In [249]:
```
#Load the input data into HDFS and make sure the output directory is cl
#!bin/hdfs dfs -put Consumer_Complaints.csv
!bin/hdfs dfs -rm -r hw_3_1_final_output
```

```
16/02/03 00:24:19 WARN util.NativeCodeLoader: Unable to load native-
hadoop library for your platform... using builtin-java classes where
applicable
16/02/03 00:24:20 INFO fs.TrashPolicyDefault: Namenode trash configu
ration: Deletion interval = 0 minutes, Emptier interval = 0 minutes.
Deleted hw_3_1_final_output
```

In [250]:
```
%%bash
#Run the job
bin/hadoop jar share/hadoop/tools/lib/hadoop-streaming-2.6.3.jar \
-file ./mapper.py    -mapper ./mapper.py \
-file ./reducer.py   -reducer ./reducer.py \
-input /user/nicholashamlin/Consumer_Complaints.csv \
-output /user/nicholashamlin/hw_3_1_final_output
```

```
                Map output bytes=4878322
                Map output materialized bytes=5504160
                Input split bytes=234
                Combine input records=0
                Combine output records=0
                Reduce input groups=10
                Reduce shuffle bytes=5504160
                Reduce input records=312913
                Reduce output records=312913
                Spilled Records=625826
                Shuffled Maps =2
                Failed Shuffles=0
                Merged Map outputs=2
                GC time elapsed (ms)=249
                CPU time spent (ms)=0
                Physical memory (bytes) snapshot=0
                Virtual memory (bytes) snapshot=0
                Total committed heap usage (bytes)=547356672
        Debt
                Total=44372
```

In the job results above, we can tell based on the counter results that there are **44372 debt records, 125752 mortgage records, and 187161 other records.**

HW 3.2 Analyze the performance of your Mappers, Combiners and Reducers using Counters

Part A

For this brief study the Input file will be one record (the next line only): foo foo quux labs foo bar quux

Perform a word count analysis of this single record dataset using a Mapper and Reducer based WordCount (i.e., no combiners are used here) using user defined Counters to count up how many time the mapper and reducer are called. What is the value of your user defined Mapper Counter, and Reducer Counter after completing this word count job. The answer should be 1 and 4 respectively. Please explain.

In [ ]:
```
#Create a test file that we can use to test our code
! echo "foo foo quux labs foo bar quux" > testfile.txt
```

HW 3.2 Part A - Mapper and Reducer

For this section, as well as the subsequent ones, we've included two counters. One that increments when the file itself is called ("script calls") and one that increments when each file prints a line ("line calls"). While it's useful to be able to see the contrasts between the two, we're mainly interested in the script calls counter for the purposes of this question.

In [251]:
```
%%writefile mapper.py
#!/usr/bin/python

#HW 2.2 - Mapper Function Code
import sys
count = 0 #Running total of occurrances for the chosen word
sys.stderr.write("reporter:counter:Mapper,Script Calls,1\n")
for line in sys.stdin:
    sys.stderr.write("reporter:counter:Mapper,Line Calls,1\n")
    line=line.strip()
    words=line.split()
    for word in words:
        print word+'\t1'
```

```
Overwriting mapper.py
```

```
In [252]: %%writefile reducer.py
          #!/usr/bin/python

          #HW 2.2 - Reducer Function Code
          import sys
          current_word=''
          count = 0 #Running total of occurrances for the chosen word

          #Increment script call counter once when the file runs
          sys.stderr.write("reporter:counter:Reducer,Script Calls,1\n")
          for line in sys.stdin:
              line=line.strip().split('\t') #Parse line into a list of fields
              word,sub_count=line
              if current_word==word:
                  count+=int(sub_count) #Extract chunk count from the second fiel
              else:
                  if current_word:

                      #Increment line call counter whenever we emit a record
                      sys.stderr.write("reporter:counter:Reducer,Line Calls,1\n")
                      print current_word+'\t'+str(count)
                  current_word=word
                  count=int(sub_count)

          #Make sure to emit final record and increment counter accordingly.
          if current_word:
              sys.stderr.write("reporter:counter:Reducer,Line Calls,1\n")
              print current_word+'\t'+str(count)
```

```
Overwriting reducer.py
```

```
In [253]: #Load the input data into HDFS and make sure the output directory is cl
          #!bin/hdfs dfs -put testfile.txt
          !bin/hdfs dfs -rm -r hw_3_2_a_output
```

```
16/02/03 00:25:10 WARN util.NativeCodeLoader: Unable to load native-
hadoop library for your platform... using builtin-java classes where
applicable
16/02/03 00:25:10 INFO fs.TrashPolicyDefault: Namenode trash configu
ration: Deletion interval = 0 minutes, Emptier interval = 0 minutes.
Deleted hw_3_2_a_output
```

In [254]:
```
%%bash
#Run the job in Hadoop using 4 reducers!
bin/hadoop jar share/hadoop/tools/lib/hadoop-streaming-2.6.3.jar \
-D mapred.map.tasks=1 \
-D mapred.reduce.tasks=4 \
-file ./mapper.py    -mapper ./mapper.py \
-file ./reducer.py   -reducer ./reducer.py \
-input /user/nicholashamlin/testfile.txt -output /user/nicholashamlin/h
```

```
packageJobJar: [./mapper.py, ./reducer.py, /var/folders/rz/drh189k95
919thyy3gs3tq400000gn/T/hadoop-unjar7165969867792931402/] [] /var/fo
lders/rz/drh189k95919thyy3gs3tq400000gn/T/streamjob88753533770334642
28.jar tmpDir=null
```

```
16/02/03 00:25:15 WARN streaming.StreamJob: -file option is deprecat
ed, please use generic option -files instead.
16/02/03 00:25:15 WARN util.NativeCodeLoader: Unable to load native-
hadoop library for your platform... using builtin-java classes where
applicable
16/02/03 00:25:16 INFO client.RMProxy: Connecting to ResourceManager
at /0.0.0.0:8032
16/02/03 00:25:16 INFO client.RMProxy: Connecting to ResourceManager
at /0.0.0.0:8032
16/02/03 00:25:17 INFO mapred.FileInputFormat: Total input paths to
process : 1
16/02/03 00:25:17 INFO mapreduce.JobSubmitter: number of splits:1
16/02/03 00:25:17 INFO Configuration.deprecation: mapred.map.tasks i
s deprecated. Instead, use mapreduce.job.maps
16/02/03 00:25:17 INFO Configuration.deprecation: mapred.reduce.task
```

Sure enough, with a single mapper and four reducers, we are able to see the corresponding values counted in the Script Calls counters in the Hadoop output above.

Part B

Please use mulitple mappers and reducers for these jobs (at least 2 mappers and 2 reducers). Perform a word count analysis of the Issue column of the Consumer Complaints Dataset using a Mapper and Reducer based WordCount (i.e., no combiners used anywhere) using user defined Counters to count up how many time the mapper and reducer are called. What is the value of your user defined Mapper Counter, and Reducer Counter after completing your word count job.

In [255]:
```
%%writefile mapper.py
#!/usr/bin/python

#HW 3.2.C - Mapper Function Code
import sys
import re
from csv import reader
WORD_RE = re.compile(r"[\w']+")

count = 0 #Running total of occurrances for the chosen word
sys.stderr.write("reporter:counter:Mapper,Script Count,1\n")
for line in reader(sys.stdin):
    sys.stderr.write("reporter:counter:Mapper,Line Count,1\n")
    try:
        int(line[0]) #check if the ID field is an integer, skip the rec
    except ValueError:
        continue

    words = re.findall(WORD_RE, line[3])
    for word in words:
        print word.lower()+'\t1'
```

```
Overwriting mapper.py
```

In [256]:
```
%%writefile reducer.py
#!/usr/bin/python

#HW 2.2 - Reducer Function Code
import sys
current_word=''
count = 0 #Running total of occurrances for the chosen word
sys.stderr.write("reporter:counter:Reducer,Script Count,1\n")
for line in sys.stdin:
    line=line.strip().split('\t') #Parse line into a list of fields
    word,sub_count=line
    if current_word==word:
        count+=int(sub_count) #Extract chunk count from the second fiel
    else:
        if current_word:
            sys.stderr.write("reporter:counter:Reducer,Line Count,1\n")
            print current_word+'\t'+str(count)
        current_word=word
        count=int(sub_count)
if current_word:
    sys.stderr.write("reporter:counter:Reducer,Line Count,1\n")
    print current_word+'\t'+str(count)
```

```
Overwriting reducer.py
```

In [257]:
```
#Make sure the output directory is clear
!bin/hdfs dfs -rm -r hw_3_2_b_output
```

```
16/02/03 00:25:56 WARN util.NativeCodeLoader: Unable to load native-
hadoop library for your platform... using builtin-java classes where
applicable
16/02/03 00:25:57 INFO fs.TrashPolicyDefault: Namenode trash configu
ration: Deletion interval = 0 minutes, Emptier interval = 0 minutes.
Deleted hw_3_2_b_output
```

In [258]:
```
%%bash
#Run the job in Hadoop, this time making sure to specify multiple mappe
bin/hadoop jar share/hadoop/tools/lib/hadoop-streaming-2.6.3.jar \
-D mapred.map.tasks=2 \
-D mapred.reduce.tasks=2 \
-file ./mapper.py    -mapper ./mapper.py \
-file ./reducer.py   -reducer ./reducer.py \
-input /user/nicholashamlin/Consumer_Complaints.csv \
-output /user/nicholashamlin/hw_3_2_b_output
```

```
packageJobJar: [./mapper.py, ./reducer.py, /var/folders/rz/drh189k95
919thyy3gs3tq400000gn/T/hadoop-unjar2970440139442435838/] [] /var/fo
lders/rz/drh189k95919thyy3gs3tq400000gn/T/streamjob47007364609100431
68.jar tmpDir=null

16/02/03 00:26:01 WARN streaming.StreamJob: -file option is deprecat
ed, please use generic option -files instead.
16/02/03 00:26:02 WARN util.NativeCodeLoader: Unable to load native-
hadoop library for your platform... using builtin-java classes where
applicable
16/02/03 00:26:02 INFO client.RMProxy: Connecting to ResourceManager
at /0.0.0.0:8032
16/02/03 00:26:03 INFO client.RMProxy: Connecting to ResourceManager
at /0.0.0.0:8032
16/02/03 00:26:03 INFO mapred.FileInputFormat: Total input paths to
process : 1
16/02/03 00:26:03 INFO mapreduce.JobSubmitter: number of splits:2
16/02/03 00:26:03 INFO Configuration.deprecation: mapred.map.tasks i
s deprecated. Instead, use mapreduce.job.maps
16/02/03 00:26:03 INFO Configuration.deprecation: mapred.reduce.task
```

In [259]:
```
# Examine the output of the job in HDFS and print some of the results t
! echo "HW 3.2 PART B RESULTS:"
! echo "10 First Results:"
!bin/hdfs dfs -cat hw_3_2_b_output/* | head -10
```

```
HW 3.2 PART B RESULTS:
10 First Results:
16/02/03 00:27:00 WARN util.NativeCodeLoader: Unable to load native-
hadoop library for your platform... using builtin-java classes where
applicable
a       3503
account 57448
acct    163
an      2964
and     16448
applied 139
apr     3431
arbitration     168
available       274
bankruptcy      222
cat: Unable to write to output stream.
```

In the two cells above, we can see that the job has run successfully, and that, as we'd expect, the script call counters for both the mapper and reducer correspond to the number of tasks we chose for the job (2 each).

Part C

Perform a word count analysis of the Issue column of the Consumer Complaints Dataset using a Mapper, Reducer, and standalone combiner (i.e., not an in-memory combiner) based WordCount using user defined Counters to count up how many time the mapper, combiner, reducer are called. What is the value of your user defined Mapper Counter, and Reducer Counter after completing your word count job.

Here, we use exactly the same mapper and reducer as the previous job, but now we add a combiner. This combiner uses the same logic as the reducer, and is only separated into its own file so we can use it to iterate on its own counter. If we didn't care about this distinction, we could simply point the job to use the reducer as the combiner and the collective work would be tracked under the single counter.

In [260]:
```
%%writefile mapper.py
#!/usr/bin/python

#HW 3.2.C - Mapper Function Code
import sys
import re
from csv import reader
WORD_RE = re.compile(r"[\w']+")

count = 0 #Running total of occurrances for the chosen word
sys.stderr.write("reporter:counter:Mapper,Script Count,1\n")
for line in reader(sys.stdin):
    sys.stderr.write("reporter:counter:Mapper,Line Count,1\n")
    try:
        int(line[0]) #check if the ID field is an integer, skip the rec
    except ValueError:
        continue

    words = re.findall(WORD_RE, line[3])
    for word in words:
        print word.lower()+'\t1'
```

```
Overwriting mapper.py
```

In [261]:
```
%%writefile combiner.py
#!/usr/bin/python

#HW 2.2 - Combiner Function Code (same as reducer)
import sys
current_word=''
count = 0 #Running total of occurrances for the chosen word
sys.stderr.write("reporter:counter:Combiner,Script Count,1\n")
for line in sys.stdin:
    line=line.strip().split('\t') #Parse line into a list of fields
    word,sub_count=line
    if current_word==word:
        count+=int(sub_count) #Extract chunk count from the second fiel
    else:
        if current_word:
            sys.stderr.write("reporter:counter:Combiner,Line Count,1\n"
            print current_word+'\t'+str(count)
        current_word=word
        count=int(sub_count)
if current_word:
    sys.stderr.write("reporter:counter:Combiner,Line Count,1\n")
    print current_word+'\t'+str(count)
```

```
Overwriting combiner.py
```

In [262]:
```
%%writefile reducer.py
#!/usr/bin/python

#HW 2.2 - Reducer Function Code
import sys
current_word=''
count = 0 #Running total of occurrances for the chosen word
sys.stderr.write("reporter:counter:Reducer,Script Count,1\n")
for line in sys.stdin:
    line=line.strip().split('\t') #Parse line into a list of fields
    word,sub_count=line
    if current_word==word:
        count+=int(sub_count) #Extract chunk count from the second fiel
    else:
        if current_word:
            sys.stderr.write("reporter:counter:Reducer,Line Count,1\n")
            print current_word+'\t'+str(count)
        current_word=word
        count=int(sub_count)
if current_word:
    sys.stderr.write("reporter:counter:Reducer,Line Count,1\n")
    print current_word+'\t'+str(count)
```

```
Overwriting reducer.py
```

In [263]:
```
#Make sure combiner is executable and the HDFS output directory is clea
#!chmod +x ./combiner.py
!bin/hdfs dfs -rm -r hw_3_2_c_output
```

```
16/02/03 00:27:34 WARN util.NativeCodeLoader: Unable to load native-
hadoop library for your platform... using builtin-java classes where
applicable
16/02/03 00:27:34 INFO fs.TrashPolicyDefault: Namenode trash configu
ration: Deletion interval = 0 minutes, Emptier interval = 0 minutes.
Deleted hw_3_2_c_output
```

```
In [264]: %%bash
#Run the word count job in Hadoop with a single reducer
bin/hadoop jar share/hadoop/tools/lib/hadoop-streaming-2.6.3.jar \
-D mapred.map.tasks=2 \
-D mapred.reduce.tasks=2 \
-file ./mapper.py    -mapper ./mapper.py \
-file ./reducer.py   -reducer ./reducer.py \
-file ./combiner.py   -combiner ./combiner.py \
-input /user/nicholashamlin/Consumer_Complaints.csv \
-output /user/nicholashamlin/hw_3_2_c_output
```

```
packageJobJar: [./mapper.py, ./reducer.py, ./combiner.py, /var/folde
rs/rz/drh189k95919thyy3gs3tq400000gn/T/hadoop-unjar67160476810321236
1/] [] /var/folders/rz/drh189k95919thyy3gs3tq400000gn/T/streamjob293
1200418503877332.jar tmpDir=null

16/02/03 00:27:38 WARN streaming.StreamJob: -file option is deprecat
ed, please use generic option -files instead.
16/02/03 00:27:39 WARN util.NativeCodeLoader: Unable to load native-
hadoop library for your platform... using builtin-java classes where
applicable
16/02/03 00:27:39 INFO client.RMProxy: Connecting to ResourceManager
at /0.0.0.0:8032
16/02/03 00:27:40 INFO client.RMProxy: Connecting to ResourceManager
at /0.0.0.0:8032
16/02/03 00:27:40 INFO mapred.FileInputFormat: Total input paths to
process : 1
16/02/03 00:27:40 INFO mapreduce.JobSubmitter: number of splits:2
16/02/03 00:27:40 INFO Configuration.deprecation: mapred.map.tasks i
s deprecated. Instead, use mapreduce.job.maps
16/02/03 00:27:40 INFO Configuration.deprecation: mapred.reduce.task
```

```
In [265]: # Examine the output of the job in HDFS and print the results
! echo "HW 3.2 PART C RESULTS:"
! echo "10 First Results:"
!bin/hdfs dfs -cat hw_3_2_c_output/* | head -10
```

```
HW 3.2 PART C RESULTS:
10 First Results:
16/02/03 00:30:26 WARN util.NativeCodeLoader: Unable to load native-
hadoop library for your platform... using builtin-java classes where
applicable
a       3503
account 57448
acct    163
an      2964
and     16448
applied 139
apr     3431
arbitration     168
available       274
bankruptcy      222
cat: Unable to write to output stream.
```

This time, when we add the combiner, we see that it runs four times, in addition to the two map and reduce tasks. Had we used the reducer as a combiner, we'd have seen 2 map tasks and 6 reduce tasks.

Part D

Using a single reducer: What are the top 50 most frequent terms in your word count analysis? Present the top 50 terms and their frequency and their relative frequency. Present the top 50 terms and their frequency and their relative frequency. If there are ties please sort the tokens in alphanumeric/string order. Present bottom 10 tokens (least frequent items).

HW 3.2 Part D - Mapper and Reducer

This code works very similarly to the word count jobs we ran in the previous homework, with the addition of the counters. Since we need to produce both raw frequencies and relative frequencies for each word, we need to use order inversion to make sure that the total word count is available to the reducer before it begins iterating through the individual words. We can accomplish this by taking advantage of the Hadoop shuffle and assigning the total a special key that will be automatically sorted to the top before the data is passed to the reducer. Thankfully, we only have one reducer, so no fancy partioning is necessary to make this work.

```
In [266]: %%writefile mapper.py
          #!/usr/bin/python

          #HW 3.2.D - Mapper Function Code
          import sys
          import re
          from csv import reader
          WORD_RE = re.compile(r"[\w']+")

          total_words=0
          count = 0 #Running total of occurrances for the chosen word
          sys.stderr.write("reporter:counter:Mapper,Script Count,1\n")
          for line in reader(sys.stdin):
              sys.stderr.write("reporter:counter:Mapper,Line Count,1\n")
              try:
                  int(line[0]) #check if the ID field is an integer, skip the rec
              except ValueError:
                  continue

              words = re.findall(WORD_RE, line[3])

              for word in words:
                  print word.lower()+'\t1' #emit one record for each word
                  total_words+=1

          #emit overall total with special key for order inversion
          print '**Total\t'+str(total_words)

          Overwriting mapper.py
```

```
In [267]: %%writefile reducer.py
          #!/usr/bin/python

          #HW 3.2.D - Reducer Function Code
          from __future__ import division
          import sys
          current_word=''
          count = 0 #Running total of occurrances for the chosen word
          number_of_words=0

          sys.stderr.write("reporter:counter:Reducer,Script Count,1\n")
          for line in sys.stdin:
              line=line.strip().split('\t') #Parse line into a list of fields
              word,sub_count=line
              sub_count=int(sub_count)

              #Extract total number of words from first record in the stream (tha
              if word=='**Total':
                  number_of_words=sub_count
                  continue

              if current_word==word:
                  count+=sub_count #Extract chunk count from the second field of
              else:
                  if current_word:
                      sys.stderr.write("reporter:counter:Reducer,Line Count,1\n")
                      print current_word+'\t'+str(count)+'\t'+str(count/number_of
                  current_word=word
                  count=int(sub_count)

          #Don't forget to emit final record
          if current_word:
              sys.stderr.write("reporter:counter:Reducer,Line Count,1\n")
              print current_word+'\t'+str(count)+'\t'+str(count/number_of_words)
```

```
Overwriting reducer.py
```

```
In [268]: #Load the input data into HDFS and make sure the output directory is cl
          #!bin/hdfs dfs -put testfile.txt
          !bin/hdfs dfs -rm -r hw_3_2_d_tmp_output
```

```
16/02/03 00:30:37 WARN util.NativeCodeLoader: Unable to load native-
hadoop library for your platform... using builtin-java classes where
applicable
16/02/03 00:30:38 INFO fs.TrashPolicyDefault: Namenode trash configu
ration: Deletion interval = 0 minutes, Emptier interval = 0 minutes.
Deleted hw_3_2_d_tmp_output
```

In [269]:
```
%%bash
#Run the word count job in Hadoop with a single reducer
bin/hadoop jar share/hadoop/tools/lib/hadoop-streaming-2.6.3.jar \
-D mapred.map.tasks=2 \
-D mapred.reduce.tasks=1 \
-file ./mapper.py    -mapper ./mapper.py \
-file ./reducer.py   -reducer ./reducer.py \
-input /user/nicholashamlin/Consumer_Complaints.csv -output /user/nicho
```

```
packageJobJar: [./mapper.py, ./reducer.py, /var/folders/rz/drh189k95
919thyy3gs3tq400000gn/T/hadoop-unjar3877860112520264432/] [] /var/fo
lders/rz/drh189k95919thyy3gs3tq400000gn/T/streamjob60063468647955573
54.jar tmpDir=null

16/02/03 00:30:41 WARN streaming.StreamJob: -file option is deprecat
ed, please use generic option -files instead.
16/02/03 00:30:41 WARN util.NativeCodeLoader: Unable to load native-
hadoop library for your platform... using builtin-java classes where
applicable
16/02/03 00:30:43 INFO client.RMProxy: Connecting to ResourceManager
at /0.0.0.0:8032
16/02/03 00:30:43 INFO client.RMProxy: Connecting to ResourceManager
at /0.0.0.0:8032
16/02/03 00:30:44 INFO mapred.FileInputFormat: Total input paths to
process : 1
16/02/03 00:30:44 INFO mapreduce.JobSubmitter: number of splits:2
16/02/03 00:30:44 INFO Configuration.deprecation: mapred.map.tasks i
s deprecated. Instead, use mapreduce.job.maps
16/02/03 00:30:44 INFO Configuration.deprecation: mapred.reduce.task
```

HW 3.2.D - Sorting via the Hadoop Shuffle using identity mapper/reducers

We can use the secondary sort functionality available in Hadoop to handle the logic associated with sorting our results. As such, we just need an identity mapper and reducer to pass the records through the shuffle. Since we'll be using this function frequently through this assignment, we'll define it once here and recycle it for subsequent sorting tasks that use this same approach.

In [270]:
```
%%writefile identity.py
#!/usr/bin/python

#HW 3.X - Identity Mapper/Reducer Function Code
import sys
for line in sys.stdin:
    print line.strip()
```

```
Overwriting identity.py
```

In [228]:
```
#make sure it's executable!
!chmod +x identity.py
```

In [271]:
```
#Load the input data into HDFS and make sure the output directory is cl
#!bin/hdfs dfs -put testfile.txt
!bin/hdfs dfs -rm -r hw_3_2_d_final_output
```

```
16/02/03 00:32:05 WARN util.NativeCodeLoader: Unable to load native-
hadoop library for your platform... using builtin-java classes where
applicable
16/02/03 00:32:06 INFO fs.TrashPolicyDefault: Namenode trash configu
ration: Deletion interval = 0 minutes, Emptier interval = 0 minutes.
Deleted hw_3_2_d_final_output
```

In [272]:
```
%%bash
#Run the sorting job using the output of the previous data in Hadoop wi
bin/hadoop jar share/hadoop/tools/lib/hadoop-streaming-2.6.3.jar \
-D mapred.map.tasks=2 \
-D mapred.reduce.tasks=1 \
-D stream.num.map.output.key.fields=2 \
-D mapred.output.key.comparator.class=org.apache.hadoop.mapred.lib.KeyF
-D  mapred.text.key.comparator.options='-k2,2nr' \
-file ./identity.py    -mapper ./identity.py \
 -reducer ./identity.py \
-input /user/nicholashamlin/hw_3_2_d_tmp_output -output /user/nicholash
```

```
packageJobJar: [./identity.py, /var/folders/rz/drh189k95919thyy3gs3t
q400000gn/T/hadoop-unjar4836867262158006418/] [] /var/folders/rz/drh
189k95919thyy3gs3tq400000gn/T/streamjob4474839850733033311.jar tmpDi
r=null

16/02/03 00:32:09 WARN streaming.StreamJob: -file option is deprecat
ed, please use generic option -files instead.
16/02/03 00:32:09 WARN util.NativeCodeLoader: Unable to load native-
hadoop library for your platform... using builtin-java classes where
applicable
16/02/03 00:32:10 INFO client.RMProxy: Connecting to ResourceManager
at /0.0.0.0:8032
16/02/03 00:32:10 INFO client.RMProxy: Connecting to ResourceManager
at /0.0.0.0:8032
16/02/03 00:32:11 INFO mapred.FileInputFormat: Total input paths to
process : 1
16/02/03 00:32:11 INFO mapreduce.JobSubmitter: number of splits:2
16/02/03 00:32:11 INFO Configuration.deprecation: mapred.map.tasks i
s deprecated. Instead, use mapreduce.job.maps
16/02/03 00:32:11 INFO Configuration.deprecation: mapred.text.key.co
```

In [273]:
```
# Examine the output of the job in HDFS and print the results
! echo "HW 3.2 PART D RESULTS:"
! echo "50 Most Common Words:"
! echo "Word | Frequency | Relative Frequency"
!bin/hdfs dfs -cat hw_3_2_d_final_output/* | head -50
! echo "===================================="
! echo "10 Least Common Words:"
! echo "Word | Frequency | Relative Frequency"
!bin/hdfs dfs -cat hw_3_2_d_final_output/* | tail -10
```

```
HW 3.2 PART D RESULTS:
50 Most Common Words:
Word | Frequency | Relative Frequency
16/02/03 00:33:02 WARN util.NativeCodeLoader: Unable to load native-
hadoop library for your platform... using builtin-java classes where
applicable
loan    119630  0.179490411074
collection      72394   0.108618480476
foreclosure     70487   0.105757256586
modification    70487   0.105757256586
account 57448   0.0861938070332
credit  55251   0.0828974730607
or      40508   0.0607773766763
payments        39993   0.0600046811843
servicing       36767   0.0551644566075
escrow  36767   0.0551644566075
report  34903   0.0523677490405
incorrect       29133   0.0437105587714
information     29069   0.0436145344772
on      29069   0.0436145344772
debt    27874   0.04182158086
closing 19000   0.0285072123247
not     18477   0.027722513796
owed    17972   0.0269648221
cont'd  17972   0.0269648221
attempts        17972   0.0269648221
collect 17972   0.0269648221
and     16448   0.0246782435956
opening 16205   0.0243136513538
management      16205   0.0243136513538
of      13983   0.0209798078914
my      10731   0.0161005734451
withdrawals     10555   0.0158365066362
deposits        10555   0.0158365066362
problems        9484    0.0142296000888
application     8868    0.0133053662577
communication   8671    0.0130097914772
tactics 8671    0.0130097914772
mortgage        8625    0.0129407740158
originator      8625    0.0129407740158
broker  8625    0.0129407740158
to      8401    0.0126046889863
unable  8178    0.0122701043364
billing 8158    0.0122400967445
other   7886    0.0118319934944
verification    7655    0.0114854058077
disclosure      7655    0.0114854058077
disputes        6938    0.0104096336373
reporting       6559    0.00984098977041
lease   6337    0.00950790550009
the     6248    0.00937437171604
low     5663    0.00849664965236
by      5663    0.00849664965236
being   5663    0.00849664965236
```

```
funds   5663    0.00849664965236
caused  5663    0.00849664965236
===================================
10 Least Common Words:
Word | Frequency | Relative Frequency
16/02/03 00:33:04 WARN util.NativeCodeLoader: Unable to load native-
hadoop library for your platform... using builtin-java classes where
applicable
apply   118     0.000177044792332
amount  98      0.000147037200412
credited        92      0.000138034922835
payment 92      0.000138034922835
convenience     75      0.000112528469703
checks  75      0.000112528469703
amt     71      0.000106526951319
day     71      0.000106526951319
disclosures     64      9.60242941464e-05
missing 64      9.60242941464e-05
```

HW3.3. Shopping Cart Analysis

Product Recommendations: The action or practice of selling additional products or services to existing customers is called cross-selling. Giving product recommendation is one of the examples of cross-selling that are frequently used by online retailers. One simple method to give product recommendations is to recommend products that are frequently browsed together by the customers.

For this homework use the online browsing behavior dataset (https://www.dropbox.com/s/zlfyiwa70poqg74/ProductPurchaseData.txt?dl=0)

Each line in this dataset represents a browsing session of a customer. On each line, each string of 8 characters represents the id of an item browsed during that session. The items are separated by spaces.

Do some exploratory data analysis of this dataset. How many unique items are available from this supplier?

Using a single reducer: Report your findings such as number of unique products; largest basket; report the top 50 most frequently purchased items, their frequency, and their relative frequency (break ties by sorting the products alphabetical order) etc. using Hadoop Map-Reduce.

HW 3.3 Mapper and Reducer #1

The first mapreduce job does the majority of the work. The mapper emits one space-delimited row with the format {product_id cart_id 1 number_of_products_in_cart] where cart_id is a auto incremented identifier for the row to make the largest cart easy to find at the end. By passing the number of products in the cart along with each product, we can track the largest cart in the reducer as we iterate through all the mapper output. In addition, we use an order-inversion pattern to emit the total number of all products seen as the first record(s) that the reducer sees. This enables lazy calculation of product relative frequencies without the need for storing many intermediate values in memory.

In [274]:
```
%%writefile mapper.py
#!/usr/bin/python

#HW 3.3 - Mapper Function Code
import sys
count = 0 #Running total of occurrances for the chosen product
product_count=0
cart_id=1 #The carts don't have IDs of their own, but we can make our c
for line in sys.stdin:
    line=line.strip()
    products=line.split() #split on whitespace

    for product in products:
        product_count+=1
        print product+' '+str(cart_id)+' 1 '+str(len(products)) #emit c
    cart_id+=1

#Emit total with special key for order inversion
print '**Total '+'0'+' '+str(product_count)+' 0' #emit total number of
```

```
Overwriting mapper.py
```

In [275]:
```
%%writefile reducer.py
#!/usr/bin/python

#HW 3.3 - Reducer Function Code
from __future__ import division
import sys
current_product=None
count = 0 #Running total of occurrances for the chosen product
largest_basket_id=0
largest_basket_size=0
unique_products=0
total_product_count=0

for line in sys.stdin:
    #Parse line into fields
    product,cart_id,sub_count,cart_total=line.strip().split(' ')
    sub_count=int(sub_count)
    cart_total=int(cart_total)

    if product=='**Total': #Extract total products for order inversion
        total_product_count+=sub_count
        continue

    #If we find a cart that's bigger than any we've seen so far, recorc
    if cart_total>largest_basket_size:
        largest_basket_size=cart_total
        largest_basket_id=cart_id

    if current_product==product:
        count+=int(sub_count)
    else:
        if current_product and current_product!='**Total':
            print current_product+'\t'+str(count)+'\t'+str(count/total_
            unique_products+=1
        current_product=product
        count=int(sub_count)
if current_product:
    print current_product+'\t'+str(count)+'\t'+str(count/total_product_
    unique_products+=1

#Print aggregated stats separately with special key to make them easy t
print '*Largest Cart\t'+str(largest_basket_id)+'\t'+str(largest_basket_
print '*Unique Products'+'\t'+str(unique_products)
```

```
Overwriting reducer.py
```

HW 3.3 - Running the jobs

In [276]:
```
### Make sure data is available and 1st job output directory is clear i
#!bin/hdfs dfs -put ProductPurchaseData.txt
!bin/hdfs dfs -rm -r hw_3_3_tmp_output
```

```
16/02/03 00:33:18 WARN util.NativeCodeLoader: Unable to load native-
hadoop library for your platform... using builtin-java classes where
applicable
16/02/03 00:33:19 INFO fs.TrashPolicyDefault: Namenode trash configu
ration: Deletion interval = 0 minutes, Emptier interval = 0 minutes.
Deleted hw_3_3_tmp_output
```

In [277]:
```
%%bash
#Run the word count job in Hadoop with a single reducer, as per the ins
bin/hadoop jar share/hadoop/tools/lib/hadoop-streaming-2.6.3.jar \
-D mapred.map.tasks=2 \
-D mapred.reduce.tasks=1 \
-file ./mapper.py    -mapper ./mapper.py \
-file ./reducer.py   -reducer ./reducer.py \
-input /user/nicholashamlin/ProductPurchaseData.txt -output /user/nicho
```

```
packageJobJar: [./mapper.py, ./reducer.py, /var/folders/rz/drh189k95
919thyy3gs3tq400000gn/T/hadoop-unjar2177335998639084698/] [] /var/fo
lders/rz/drh189k95919thyy3gs3tq400000gn/T/streamjob57202229079153574
2.jar tmpDir=null
```

```
16/02/03 00:33:21 WARN streaming.StreamJob: -file option is deprecat
ed, please use generic option -files instead.
16/02/03 00:33:21 WARN util.NativeCodeLoader: Unable to load native-
hadoop library for your platform... using builtin-java classes where
applicable
16/02/03 00:33:22 INFO client.RMProxy: Connecting to ResourceManager
at /0.0.0.0:8032
16/02/03 00:33:23 INFO client.RMProxy: Connecting to ResourceManager
at /0.0.0.0:8032
16/02/03 00:33:23 INFO mapred.FileInputFormat: Total input paths to
process : 1
16/02/03 00:33:23 INFO mapreduce.JobSubmitter: number of splits:2
16/02/03 00:33:23 INFO Configuration.deprecation: mapred.map.tasks i
s deprecated. Instead, use mapreduce.job.maps
16/02/03 00:33:23 INFO Configuration.deprecation: mapred.reduce.task
```

HW 3.3 Mapper and Reducer #2

As in previous problems, we run a second job focused solely on sorting the output of the first job. We can use the identity mapper/reducer from earlier and tackle the entire sort via the shuffle using secondary sort keys.

```
In [278]: #Make sure the destination directory for the second job is clear
          !bin/hdfs dfs -rm -r hw_3_3_final_output
```

```
16/02/03 00:34:59 WARN util.NativeCodeLoader: Unable to load native-
hadoop library for your platform... using builtin-java classes where
applicable
16/02/03 00:35:00 INFO fs.TrashPolicyDefault: Namenode trash configu
ration: Deletion interval = 0 minutes, Emptier interval = 0 minutes.
Deleted hw_3_3_final_output
```

```
In [279]: %%bash
          #Sort the results in the shuffle, using identity mapper/reducers
          time bin/hadoop jar share/hadoop/tools/lib/hadoop-streaming-2.6.3.jar \
          -D mapred.map.tasks=2 \
          -D mapred.reduce.tasks=1 \
          -D stream.num.map.output.key.fields=2 \
          -D mapred.output.key.comparator.class=org.apache.hadoop.mapred.lib.KeyF
          -D mapred.text.key.comparator.options='-k2,2nr -k1,1' \
          -file ./identity.py    -mapper ./identity.py \
          -reducer ./identity.py \
          -input /user/nicholashamlin/hw_3_3_tmp_output -output /user/nicholasham
```

```
packageJobJar: [./identity.py, /var/folders/rz/drh189k95919thyy3gs3t
q400000gn/T/hadoop-unjar918449771401492720/] [] /var/folders/rz/drh1
89k95919thyy3gs3tq400000gn/T/streamjob2493525553787362292.jar tmpDir=
null

16/02/03 00:35:02 WARN streaming.StreamJob: -file option is deprecat
ed, please use generic option -files instead.
16/02/03 00:35:02 WARN util.NativeCodeLoader: Unable to load native-
hadoop library for your platform... using builtin-java classes where
applicable
16/02/03 00:35:03 INFO client.RMProxy: Connecting to ResourceManager
at /0.0.0.0:8032
16/02/03 00:35:04 INFO client.RMProxy: Connecting to ResourceManager
at /0.0.0.0:8032
16/02/03 00:35:04 INFO mapred.FileInputFormat: Total input paths to
process : 1
16/02/03 00:35:04 INFO mapreduce.JobSubmitter: number of splits:2
16/02/03 00:35:04 INFO Configuration.deprecation: mapred.map.tasks i
s deprecated. Instead, use mapreduce.job.maps
16/02/03 00:35:04 INFO Configuration.deprecation: mapred.text.key.co
```

In [280]:

```
# Examine the output of the job in HDFS and print the results
! echo "HW 3.3 RESULTS:"
! echo ""
! echo "50 Most Frequent Products (Summary Stats in first two rows):"
! echo "Product ID | Raw Frequency | Relative Frequency"
!bin/hdfs dfs -cat hw_3_3_final_output/* | head -52
```

```
HW 3.3 RESULTS:

50 Most Frequent Products (Summary Stats in first two rows):
Product ID | Raw Frequency | Relative Frequency
16/02/03 00:36:12 WARN util.NativeCodeLoader: Unable to load native-
hadoop library for your platform... using builtin-java classes where
applicable
*Unique Products        12592
*Largest Cart   6914    37
DAI62779        6667    0.0175067747831
FRO40251        3881    0.010191059387
ELE17451        3875    0.0101753040775
GRO73461        3602    0.00945843749344
SNA80324        3044    0.00799319370628
ELE32164        2851    0.0074863979161
DAI75645        2736    0.00718442114993
SNA45677        2455    0.0064465474865
FRO31317        2330    0.0061183118711
DAI85309        2293    0.00602115412894
ELE26917        2292    0.00601852824402
FRO80039        2233    0.00586360103355
GRO21487        2115    0.00555374661261
SNA99873        2083    0.00546971829507
GRO59710        2004    0.00526227338613
GRO71621        1920    0.00504169905258
FRO85978        1918    0.00503644728273
GRO30386        1840    0.00483162825872
ELE74009        1816    0.00476860702057
GRO56726        1784    0.00468457870302
DAI63921        1773    0.00465569396887
GRO46854        1756    0.00461105392517
ELE66600        1713    0.00449814087347
DAI83733        1712    0.00449551498855
FRO32293        1702    0.00446925613932
ELE66810        1697    0.0044561267147
SNA55762        1646    0.00432220658362
DAI22177        1627    0.00427231477008
FRO78087        1531    0.00402022981745
ELE99737        1516    0.0039808415436
ELE34057        1489    0.00390994265067
GRO94758        1489    0.00390994265067
FRO35904        1436    0.00377077074974
FRO53271        1420    0.00372875659097
SNA93860        1407    0.00369462008697
SNA90094        1390    0.00364998004327
GRO38814        1352    0.00355019641619
ELE56788        1345    0.00353181522173
GRO61133        1321    0.00346879398357
DAI88807        1316    0.00345566455896
ELE74482        1316    0.00345566455896
ELE59935        1311    0.00344253513434
SNA96271        1295    0.00340052097557
DAI43223        1290    0.00338739155095
ELE91337        1289    0.00338476566603
```

```
GRO15017        1275    0.0033480032771
DAI31081        1261    0.00331124088818
GRO81087        1220    0.00320357960633
DAI22896        1219    0.0032009537214
GRO85051        1214    0.00318782429679
cat: Unable to write to output stream.
```

Based on these results, we have 12592 unique products browsed. The largest session covered 37 products, and the most commonly browsed product was DAI62779, which was seen 6667 times for a relative frequency of 0.0175.

HW3.4. Pairs

Suppose we want to recommend new products to the customer based on the products they have already browsed on the online website. Write a map-reduce program to find products which are frequently browsed together. Fix the support count (cooccurence count) to s = 100 (i.e. product pairs need to occur together at least 100 times to be considered frequent) and find pairs of items (sometimes referred to itemsets of size 2 in association rule mining) that have a support count of 100 or more.

List the top 50 product pairs with corresponding support count (aka frequency), and relative frequency or support (number of records where they coccur, the number of records where they coccur/the number of baskets in the dataset) in decreasing order of support for frequent (100>count) itemsets of size 2.

Use the Pairs pattern (lecture 3) to extract these frequent itemsets of size 2. Free free to use combiners if they bring value. Instrument your code with counters for count the number of times your mapper, combiner and reducers are called.

Please output records of the following form for the top 50 pairs (itemsets of size 2):

```
item1, item2, support count, support
```

Fix the ordering of the pairs lexicographically (left to right), and break ties in support (between pairs, if any exist) by taking the first ones in lexicographically increasing order.

Report the compute time for the Pairs job. Describe the computational setup used (E.g., single computer; dual core; linux, number of mappers, number of reducers) Instrument your mapper, combiner, and reducer to count how many times each is called using Counters and report these counts.

HW 3.4 - Mapper and Reducer #1

The first job does most of the work here. Each cart is broken into a lists of products. As we iterate through the list, a pair is emitted for each combination of the current product and every subsequent product that follows. This ensures that we don't double count pairs.

Similarly, pairs are sorted lexicographically before being sent to the reducer. As before, we use order inversion to ensure that the reducers can access the overal total and compute relative frequency efficiently.

In the reducer, we take the same approach as we have with past word counts, with the key difference being that two pieces of information are compared (both products) to the current state of each iteration. If the incoming pair matches the present pair, we increment our totals. If not, we emit the record.

In [281]:
```
%%writefile mapper.py
#!/usr/bin/python

#HW 3.4 - Mapper Function Code
import sys
number_of_carts=0
sys.stderr.write("reporter:counter:Mapper,Script Count,1\n")

#Define data split for custom partitioner
group1 = "abcdefghijklm"
group2 = "nopqrstuvwxyz"

for line in sys.stdin:
    line=line.strip()
    products=line.split() #split on whitespace

    for i,product in enumerate(products):
        product_a=product

        #Only iterate through products we haven't seen yet to avoid dup
        for product_b in products[i+1:]:

            #Sort output in alphabetical order
            output=sorted([product_a,product_b])
            sys.stderr.write("reporter:counter:Mapper,Line Count,1\n")

            #Emit one pair for every result, but with extra key for cus
            if product_a[0].lower() in group1:
                print output[0]+'\t'+output[1]+'\t1\t1'
            else:
                print output[0]+'\t'+output[1]+'\t1\t2'
    number_of_carts+=1

#Output total number of carts with a special key for order-inversion pu
#Two versions will enable one record for each reducer.
print '**Total'+'\t'+'**Total'+'\t'+str(number_of_carts)+'\t1'
print '**Total'+'\t'+'**Total'+'\t'+str(number_of_carts)+'\t2'
sys.stderr.write("reporter:counter:Mapper,Line Count,1\n")
```

```
Overwriting mapper.py
```

```
In [282]: %%writefile reducer.py
          #!/usr/bin/python

          #HW 3.4 - Reducer Function Code
          from __future__ import division
          import sys

          sys.stderr.write("reporter:counter:Reducer,Script Count,1\n")

          s=100 #cutoff for "frequent"

          #We want to track two products at a time as they come in from the mappe
          current_product_a=None
          current_product_b=None
          count = 0 #Running total of occurrances for the chosen product

          number_of_carts=0

          for line in sys.stdin:
              #Parse line into fields
              product_a,product_b,support=line.strip().split('\t')[0:3]
              support=int(support)

              #Extract total products for order inversion
              if product_a=='**Total' and product_b=='**Total':
                  number_of_carts+=support
                  continue

              #Only increment counter if both products match
              if current_product_a==product_a and current_product_b==product_b:
                  count+=support
              else:
                  if current_product_a and current_product_b and current_product_
                      if count>=s: #only emit records with at least 100 pairs
                          print current_product_a+'\t'+current_product_b+'\t'+str
                          sys.stderr.write("reporter:counter:Reducer,Line Count,1
                  current_product_a=product_a
                  current_product_b=product_b
                  count=support

          #Make sure to emit final result as well
          if current_product_a and current_product_b and current_product_a!='**To
              if count>=s:
                  print current_product_a+'\t'+current_product_b+'\t'+str(count)+
                  sys.stderr.write("reporter:counter:Reducer,Line Count,1\n")
```

```
Overwriting reducer.py
```

HW 3.4 - Running the jobs

The first job is the one we're interested in tracking, since it's where we've implemented the pairs approach. To monitor how long it takes, we use the simple `time` command available in bash.

In [283]:
```
### Make sure 1st job output directory is clear in HDFS
#!bin/hdfs dfs -put purchase_test.txt
!bin/hdfs dfs -rm -r hw_3_4_tmp_output
#!bin/hdfs dfs -ls
```

```
16/02/03 00:36:26 WARN util.NativeCodeLoader: Unable to load native-
hadoop library for your platform... using builtin-java classes where
applicable
16/02/03 00:36:27 INFO fs.TrashPolicyDefault: Namenode trash configu
ration: Deletion interval = 0 minutes, Emptier interval = 0 minutes.
Deleted hw_3_4_tmp_output
```

In [284]:
```
%%bash
#Run the first job in Hadoop, making sure to time the results

time bin/hadoop jar share/hadoop/tools/lib/hadoop-streaming-2.6.3.jar \
-D mapred.map.tasks=2 \
-D mapred.reduce.tasks=2 \
-D mapreduce.partition.keypartitioner.options='-k1,2 -k4,4' \
-D stream.num.map.output.key.fields=4 \
-D mapred.output.key.comparator.class=org.apache.hadoop.mapred.lib.KeyF
-D mapred.text.key.comparator.options='-k1,1 -k2,2' \
-file ./mapper.py    -mapper ./mapper.py \
-file ./reducer.py   -reducer ./reducer.py \
-input /user/nicholashamlin/ProductPurchaseData.txt -output /user/nicho
```

```
packageJobJar: [./mapper.py, ./reducer.py, /var/folders/rz/drh189k95
919thyy3gs3tq400000gn/T/hadoop-unjar1228170282575044486/] [] /var/fol
ders/rz/drh189k95919thyy3gs3tq400000gn/T/streamjob347218737327176769
9.jar tmpDir=null

16/02/03 00:36:30 WARN streaming.StreamJob: -file option is deprecat
ed, please use generic option -files instead.
16/02/03 00:36:30 WARN util.NativeCodeLoader: Unable to load native-
hadoop library for your platform... using builtin-java classes where
applicable
16/02/03 00:36:31 INFO client.RMProxy: Connecting to ResourceManager
at /0.0.0.0:8032
16/02/03 00:36:31 INFO client.RMProxy: Connecting to ResourceManager
at /0.0.0.0:8032
16/02/03 00:36:32 INFO mapred.FileInputFormat: Total input paths to
process : 1
16/02/03 00:36:32 INFO mapreduce.JobSubmitter: number of splits:2
16/02/03 00:36:32 INFO Configuration.deprecation: mapred.map.tasks i
s deprecated. Instead, use mapreduce.job.maps
16/02/03 00:36:32 INFO Configuration.deprecation: mapred.text.key.co
```

HW 3.4 - Mapper and Reducer #2

This second job uses the same identity mapper and reducer from earlier to feed records throught the shuffle, which takes care of the sorting we need. We need to wait until the first job is completed to do this sort, otherwise we won't have calculated the overall totals for each pair.

In [288]:
```
#Make sure final output directory is clear
!bin/hdfs dfs -rm -r hw_3_4_final_output
```

```
16/02/03 00:42:32 WARN util.NativeCodeLoader: Unable to load native-
hadoop library for your platform... using builtin-java classes where
applicable
16/02/03 00:42:33 INFO fs.TrashPolicyDefault: Namenode trash configu
ration: Deletion interval = 0 minutes, Emptier interval = 0 minutes.
Deleted hw_3_4_final_output
```

In [289]:
```
%%bash
#Run the sorting job using the output of the previous data in Hadoop wi
time bin/hadoop jar share/hadoop/tools/lib/hadoop-streaming-2.6.3.jar \
-D stream.num.map.output.key.fields=3 \
-D mapred.output.key.comparator.class=org.apache.hadoop.mapred.lib.KeyF
-D mapred.text.key.comparator.options='-k 3,3nr -k 1,1 -k 2,2' \
-file ./identity.py    -mapper ./identity.py \
-reducer ./identity.py \
-input /user/nicholashamlin/hw_3_4_tmp_output -output /user/nicholasham
```

```
packageJobJar: [./identity.py, /var/folders/rz/drh189k95919thyy3gs3t
q400000gn/T/hadoop-unjar7188213226497342987/] [] /var/folders/rz/drh
189k95919thyy3gs3tq400000gn/T/streamjob7565968427572811265.jar tmpDi
r=null

16/02/03 00:42:35 WARN streaming.StreamJob: -file option is deprecat
ed, please use generic option -files instead.
16/02/03 00:42:35 WARN util.NativeCodeLoader: Unable to load native-
hadoop library for your platform... using builtin-java classes where
applicable
16/02/03 00:42:37 INFO client.RMProxy: Connecting to ResourceManager
at /0.0.0.0:8032
16/02/03 00:42:37 INFO client.RMProxy: Connecting to ResourceManager
at /0.0.0.0:8032
16/02/03 00:42:37 INFO mapred.FileInputFormat: Total input paths to
process : 2
16/02/03 00:42:37 INFO mapreduce.JobSubmitter: number of splits:2
16/02/03 00:42:37 INFO Configuration.deprecation: mapred.text.key.co
mparator.options is deprecated. Instead, use mapreduce.partition.key
comparator.options
```

In [290]:
```
# Examine the output of the job in HDFS and print the results
! echo "HW 3.4 RESULTS:"
! echo "50 Most Frequent Pairs:"
! echo "Product 1   |   Product 2 | Raw Freq. | Support "
!bin/hdfs dfs -cat hw_3_4_final_output/* | head -50
```

```
HW 3.4 RESULTS:
50 Most Frequent Pairs:
Product 1    |   Product 2 | Raw Freq. | Support
16/02/03 00:43:02 WARN util.NativeCodeLoader: Unable to load native-
hadoop library for your platform... using builtin-java classes where
applicable
DAI62779        ELE17451        1592    0.0511880646925
DAI75645        FRO40251        1254    0.0403202469374
FRO40251        GRO85051        1213    0.0390019613517
DAI62779        GRO73461        1139    0.0366226166361
DAI62779        FRO40251        1070    0.0344040384554
FRO40251        SNA80324        963     0.0309636346098
DAI62779        DAI85309        918     0.0295167357963
ELE32164        GRO59710        911     0.0292916626475
DAI62779        DAI75645        882     0.0283592167454
FRO40251        GRO73461        882     0.0283592167454
DAI62779        ELE92920        877     0.0281984502106
FRO40251        FRO92469        835     0.026848011318
DAI62779        ELE32164        832     0.0267515513971
DAI75645        GRO73461        712     0.0228931545609
DAI43223        ELE32164        711     0.022861001254
DAI62779        GRO30386        709     0.02279669464
ELE17451        FRO40251        697     0.0224108549564
DAI62779        SNA80324        662     0.0212854892126
DAI85309        ELE99737        659     0.0211890292917
DAI62779        ELE26917        650     0.020899649529
GRO21487        GRO73461        631     0.0202887366966
DAI62779        GRO71621        595     0.0191312176457
DAI75645        SNA80324        589     0.0189382978039
DAI62779        DAI83733        586     0.018841837883
ELE17451        GRO73461        580     0.0186489180412
DAI62779        GRO59710        561     0.0180380052088
DAI62779        FRO80039        550     0.0176843188322
DAI75645        ELE17451        547     0.0175878589113
DAI75645        SNA80324        541     0.0173949390695
GRO73461        SNA80324        539     0.0173306324555
DAI55148        DAI62779        526     0.016912639465
DAI43223        GRO59710        512     0.0164624931674
ELE17451        ELE32164        511     0.0164303398605
ELE32164        GRO73461        486     0.0156265071863
DAI62779        FRO78087        482     0.0154978939584
DAI85309        ELE17451        482     0.0154978939584
DAI62779        GRO94758        479     0.0154014340375
DAI62779        GRO21487        471     0.0151442075817
ELE17451        GRO30386        468     0.0150477476608
DAI62779        FRO19221        462     0.014854827819
DAI62779        SNA18336        462     0.014854827819
DAI62779        GRO46854        461     0.0148226745121
DAI43223        DAI62779        459     0.0147583678981
FRO85978        SNA95666        459     0.0147583678981
ELE92920        SNA18336        451     0.0145011414424
FRO40251        SNA80324        449     0.0144368348285
DAI88079        FRO40251        446     0.0143403749076
FRO73056        GRO44993        438     0.0140831484518
```

```
DAI62779        FRO85978        434     0.0139545352239
ELE20847        FRO40251        434     0.0139545352239
cat: Unable to write to output stream.
```

HW 3.4 - Discussion

We ran these jobs on a dual-core Macbook Pro running a local install of Hadoop 2.6.3 with 2 mappers and 2 reducers. **The main pairs job ran in 40.3 seconds** (as shown above in the output stream) and called the mapper and reducer function twice each, as we'd expect. The second simple sort job using an identity mapper/reducer ran on the default 2 mappers/1 reducer configuration in about 20 seconds.

HW3.5. Stripes

Repeat 3.4 using the stripes design pattern for finding cooccuring pairs.

Report the compute times for stripes job versus the Pairs job. Describe the computational setup used (E.g., single computer; dual core; linux, number of mappers, number of reducers)

Instrument your mapper, combiner, and reducer to count how many times each is called using Counters and report these counts. Discuss the differences in these counts between the Pairs and Stripes jobs

HW 3.5 Mapper and Reducer #1

For the stripes implementation, the mapper emits an associative array for each product that contains the list of products paired with it. In the reducer, these arrays are unpacked and summed to product the final total for each pair of products. We implement this simply in python using the Counter object, which enables us to increment the marginal product counts using similar syntax to the pairs approach and without needing to worry about locating and incrementing each individual value for each individual product.

```
In [291]: %%writefile mapper.py
#!/usr/bin/python

#HW 3.5 - Mapper #1 Function Code
import sys
sys.stderr.write("reporter:counter:Mapper,Script Count,1\n")
number_of_carts=0

#Define data split for custom partitioner
group1 = "abcdefghijklm"
group2 = "nopqrstuvwxyz"

for line in sys.stdin:
    line=line.strip()
    products=line.split() #split on whitespace
    try:
        products.sort() #This products against double-counting
    except:
        pass #don't bother sorting if there aren't any pairs
    for i,product_a in enumerate(products):
        line_output={}
        for product_b in products[i+1:]:
            try:
                line_output[product_b]+=1
            except KeyError: #If we haven't seen a product before, add
                line_output[product_b]=1

        if line_output.keys(): #only emit a record if we find more than
            if product_a[0].lower() in group1:
                print product_a+'\t1\t'+str(line_output)
            else:
                print product_a+'\t2\t'+str(line_output)
            sys.stderr.write("reporter:counter:Mapper,Line Count,1\n")
    number_of_carts+=1


print '**Total'+'\t1\t{"number_of_carts":'+str(number_of_carts)+'}'
print '**Total'+'\t2\t{"number_of_carts":'+str(number_of_carts)+'}'
sys.stderr.write("reporter:counter:Mapper,Line Count,1\n")
```

```
Overwriting mapper.py
```

```
In [292]: %%writefile reducer.py
#!/usr/bin/python

#HW 3.5 - Reducer #1 Function Code
from __future__ import division
import sys
from collections import Counter, OrderedDict
sys.stderr.write("reporter:counter:Reducer,Script Count,1\n")
s=100 #cutoff for "frequent"
current_product_dict=Counter({}) #Counters make tracking individual pro
current_product=None
count = 0 #Running total of occurrances for the chosen product

number_of_carts=0

for line in sys.stdin:
    #Parse line into fields
    product,group,product_dict=line.strip().split('\t')
    #The dict is passed from the mapper as a string, so we need to conv
    product_dict=Counter(eval(product_dict))

    if product=='**Total': #Extract total products for order inversion
        number_of_carts+=product_dict['number_of_carts']
        continue

    if current_product==product:
        #The counter is smart enough to increment keys when they exist,
        current_product_dict+=Counter(product_dict)
    else:
        if current_product and current_product!='**Total':
            #Ordering the results ensures we maintain lexicographic sor
            for i in OrderedDict(sorted(current_product_dict.items())):
                if current_product_dict[i]>=s:
                    sys.stderr.write("reporter:counter:Reducer,Line Cou
                    print current_product+'\t'+i+'\t'+str(current_produ
        current_product=product
        current_product_dict=product_dict

#Make sure to emit final row
if current_product and current_product!='**Total':
    for i in OrderedDict(sorted(current_product_dict.items())):
        if current_product_dict[i]>=s:
            sys.stderr.write("reporter:counter:Reducer,Line Count,1\n")
            print current_product+'\t'+i+'\t'+str(current_product_dict[
```

```
Overwriting reducer.py
```

HW 3.5 - Second mapper/reducer and running the job

As in our pairs implementation, we can use the same identity mapper/reducer to pass records through a Hadoop shuffle with secondary sort to enable the proper final ordering of the records from our first job.

In [293]:
```
### Make sure 1st job output directory is clear in HDFS
!bin/hdfs dfs -rm -r hw_3_5_tmp_output
#!bin/hdfs dfs -ls
```

```
16/02/03 00:43:25 WARN util.NativeCodeLoader: Unable to load native-
hadoop library for your platform... using builtin-java classes where
applicable
16/02/03 00:43:25 INFO fs.TrashPolicyDefault: Namenode trash configu
ration: Deletion interval = 0 minutes, Emptier interval = 0 minutes.
Deleted hw_3_5_tmp_output
```

```
In [294]: %%bash
#Run the primary stripes job in Hadoop
time bin/hadoop jar share/hadoop/tools/lib/hadoop-streaming-2.6.3.jar \
-D mapred.map.tasks=2 \
-D mapred.reduce.tasks=2 \
-D stream.num.map.output.key.fields=2 \
-D mapreduce.partition.keypartitioner.options='-k1,2' \
-D mapred.output.key.comparator.class=org.apache.hadoop.mapred.lib.KeyF
-D mapred.text.key.comparator.options='-k1,1' \
-file ./mapper.py    -mapper ./mapper.py \
-file ./reducer.py   -reducer ./reducer.py \
-input /user/nicholashamlin/ProductPurchaseData.txt -output /user/nicho
```

```
packageJobJar: [./mapper.py, ./reducer.py, /var/folders/rz/drh189k95
919thyy3gs3tq400000gn/T/hadoop-unjar451974825488861623/] [] /var/fol
ders/rz/drh189k95919thyy3gs3tq400000gn/T/streamjob701620698250507166
6.jar tmpDir=null
```

```
16/02/03 00:43:27 WARN streaming.StreamJob: -file option is deprecat
ed, please use generic option -files instead.
16/02/03 00:43:27 WARN util.NativeCodeLoader: Unable to load native-
hadoop library for your platform... using builtin-java classes where
applicable
16/02/03 00:43:28 INFO client.RMProxy: Connecting to ResourceManager
at /0.0.0.0:8032
16/02/03 00:43:28 INFO client.RMProxy: Connecting to ResourceManager
at /0.0.0.0:8032
16/02/03 00:43:29 INFO mapred.FileInputFormat: Total input paths to
process : 1
16/02/03 00:43:29 INFO mapreduce.JobSubmitter: number of splits:2
16/02/03 00:43:29 INFO Configuration.deprecation: mapred.map.tasks i
s deprecated. Instead, use mapreduce.job.maps
16/02/03 00:43:29 INFO Configuration.deprecation: mapred.text.key.co
mparator.options is deprecated. Instead, use mapreduce.partition.key
comparator.options
16/02/03 00:43:29 INFO Configuration.deprecation: mapred.reduce.task
s is deprecated. Instead, use mapreduce.job.reduces
16/02/03 00:43:29 INFO Configuration.deprecation: mapred.output.key.
comparator.class is deprecated. Instead, use mapreduce.job.output.ke
y.comparator.class
16/02/03 00:43:29 INFO mapreduce.JobSubmitter: Submitting tokens for
job: job_1454033924139_0124
16/02/03 00:43:29 INFO impl.YarnClientImpl: Submitted application ap
plication_1454033924139_0124
16/02/03 00:43:29 INFO mapreduce.Job: The url to track the job: htt
p://Nicholass-MacBook-Pro.local:8088/proxy/application_1454033924139
_0124/ (http://Nicholass-MacBook-Pro.local:8088/proxy/application_14
54033924139_0124/)
16/02/03 00:43:29 INFO mapreduce.Job: Running job: job_1454033924139
_0124
16/02/03 00:43:34 INFO mapreduce.Job: Job job_1454033924139_0124 run
ning in uber mode : false
16/02/03 00:43:34 INFO mapreduce.Job:  map 0% reduce 0%
16/02/03 00:43:44 INFO mapreduce.Job:  map 100% reduce 0%
```

```
16/02/03 00:43:54 INFO mapreduce.Job:  map 100% reduce 34%
16/02/03 00:43:56 INFO mapreduce.Job:  map 100% reduce 69%
16/02/03 00:43:57 INFO mapreduce.Job:  map 100% reduce 70%
16/02/03 00:43:59 INFO mapreduce.Job:  map 100% reduce 71%
16/02/03 00:44:00 INFO mapreduce.Job:  map 100% reduce 72%
16/02/03 00:44:02 INFO mapreduce.Job:  map 100% reduce 73%
16/02/03 00:44:03 INFO mapreduce.Job:  map 100% reduce 74%
16/02/03 00:44:08 INFO mapreduce.Job:  map 100% reduce 75%
16/02/03 00:44:15 INFO mapreduce.Job:  map 100% reduce 76%
16/02/03 00:44:18 INFO mapreduce.Job:  map 100% reduce 77%
16/02/03 00:44:19 INFO mapreduce.Job:  map 100% reduce 78%
16/02/03 00:44:22 INFO mapreduce.Job:  map 100% reduce 79%
16/02/03 00:44:25 INFO mapreduce.Job:  map 100% reduce 80%
16/02/03 00:44:28 INFO mapreduce.Job:  map 100% reduce 81%
16/02/03 00:44:30 INFO mapreduce.Job:  map 100% reduce 82%
16/02/03 00:44:33 INFO mapreduce.Job:  map 100% reduce 83%
16/02/03 00:44:36 INFO mapreduce.Job:  map 100% reduce 84%
16/02/03 00:44:37 INFO mapreduce.Job:  map 100% reduce 85%
16/02/03 00:44:40 INFO mapreduce.Job:  map 100% reduce 86%
16/02/03 00:44:42 INFO mapreduce.Job:  map 100% reduce 87%
16/02/03 00:44:45 INFO mapreduce.Job:  map 100% reduce 88%
16/02/03 00:44:46 INFO mapreduce.Job:  map 100% reduce 89%
16/02/03 00:44:49 INFO mapreduce.Job:  map 100% reduce 90%
16/02/03 00:44:51 INFO mapreduce.Job:  map 100% reduce 91%
16/02/03 00:44:54 INFO mapreduce.Job:  map 100% reduce 92%
16/02/03 00:44:55 INFO mapreduce.Job:  map 100% reduce 93%
16/02/03 00:44:57 INFO mapreduce.Job:  map 100% reduce 94%
16/02/03 00:44:58 INFO mapreduce.Job:  map 100% reduce 95%
16/02/03 00:45:04 INFO mapreduce.Job:  map 100% reduce 96%
16/02/03 00:45:07 INFO mapreduce.Job:  map 100% reduce 97%
16/02/03 00:45:13 INFO mapreduce.Job:  map 100% reduce 98%
16/02/03 00:45:16 INFO mapreduce.Job:  map 100% reduce 99%
16/02/03 00:45:21 INFO mapreduce.Job:  map 100% reduce 100%
16/02/03 00:45:22 INFO mapreduce.Job: Job job_1454033924139_0124 com
pleted successfully
16/02/03 00:45:22 INFO mapreduce.Job: Counters: 53
        File System Counters
                FILE: Number of bytes read=43180068
                FILE: Number of bytes written=86801358
                FILE: Number of read operations=0
                FILE: Number of large read operations=0
                FILE: Number of write operations=0
                HDFS: Number of bytes read=3462847
                HDFS: Number of bytes written=51765
                HDFS: Number of read operations=12
                HDFS: Number of large read operations=0
                HDFS: Number of write operations=4
        Job Counters
                Launched map tasks=2
                Launched reduce tasks=2
                Data-local map tasks=2
                Total time spent by all maps in occupied slots (ms)=
15730
                Total time spent by all reduces in occupied slots (m
s)=165371
                Total time spent by all map tasks (ms)=15730
                Total time spent by all reduce tasks (ms)=165371
                Total vcore-milliseconds taken by all map tasks=1573
0
                Total vcore-milliseconds taken by all reduce tasks=1
65371
                Total megabyte-milliseconds taken by all map tasks=1
6107520
                Total megabyte-milliseconds taken by all reduce task
s=169339904
        Map-Reduce Framework
                Map input records=31101
                Map output records=349727
                Map output bytes=42341631
                Map output materialized bytes=43180080
                Input split bytes=234
                Combine input records=0
                Combine output records=0
                Reduce input groups=12013
                Reduce shuffle bytes=43180080
                Reduce input records=349727
                Reduce output records=1334
                Spilled Records=699454
                Shuffled Maps =4
                Failed Shuffles=0
                Merged Map outputs=4
                GC time elapsed (ms)=360
                CPU time spent (ms)=0
                Physical memory (bytes) snapshot=0
                Virtual memory (bytes) snapshot=0
                Total committed heap usage (bytes)=680525824
        Mapper
                Line Count=349725
                Script Count=2
        Reducer
                Line Count=1334
```

In [295]:
```
#Make sure the final output directory is clear
!bin/hdfs dfs -rm -r hw_3_5_final_output
```

```
16/02/03 00:47:48 WARN util.NativeCodeLoader: Unable to load native-
hadoop library for your platform... using builtin-java classes where
applicable
16/02/03 00:47:48 INFO fs.TrashPolicyDefault: Namenode trash configu
ration: Deletion interval = 0 minutes, Emptier interval = 0 minutes.
Deleted hw_3_5_final_output
```

In [296]:
```
%%bash
#Run the sorting job with identity mapper/reducer
time bin/hadoop jar share/hadoop/tools/lib/hadoop-streaming-2.6.3.jar \
-D stream.num.map.output.key.fields=3 \
-D mapred.output.key.comparator.class=org.apache.hadoop.mapred.lib.KeyF
-D mapred.text.key.comparator.options='-k 3,3nr -k 1,1 -k 2,2' \
-file ./identity.py    -mapper ./identity.py \
-reducer ./identity.py \
-input /user/nicholashamlin/hw_3_5_tmp_output -output /user/nicholasham
```

```
packageJobJar: [./identity.py, /var/folders/rz/drh189k95919thyy3gs3t
q400000gn/T/hadoop-unjar3891535724588519065/] [] /var/folders/rz/drh
189k95919thyy3gs3tq400000gn/T/streamjob6117757244022260949.jar tmpDi
r=null

16/02/03 00:47:51 WARN streaming.StreamJob: -file option is deprecat
ed, please use generic option -files instead.
16/02/03 00:47:51 WARN util.NativeCodeLoader: Unable to load native-
hadoop library for your platform... using builtin-java classes where
applicable
16/02/03 00:47:52 INFO client.RMProxy: Connecting to ResourceManager
at /0.0.0.0:8032
16/02/03 00:47:52 INFO client.RMProxy: Connecting to ResourceManager
at /0.0.0.0:8032
16/02/03 00:47:53 INFO mapred.FileInputFormat: Total input paths to
process : 2
16/02/03 00:47:53 INFO mapreduce.JobSubmitter: number of splits:3
16/02/03 00:47:53 INFO Configuration.deprecation: mapred.text.key.co
mparator.options is deprecated. Instead, use mapreduce.partition.key
comparator.options
```

```
In [297]: # Examine the output of the job in HDFS and print the results
          ! echo "HW 3.5 RESULTS:"
          ! echo ""
          ! echo "50 Most Frequent Pairs:"
          ! echo "Product 1   |   Product 2 | Raw Freq. | Support "
          !bin/hdfs dfs -cat hw_3_5_final_output/* | head -50
```

```
HW 3.5 RESULTS:

50 Most Frequent Pairs:
Product 1    |    Product 2 | Raw Freq. | Support
16/02/03 00:49:02 WARN util.NativeCodeLoader: Unable to load native-
hadoop library for your platform... using builtin-java classes where
applicable
DAI62779        ELE17451        1592    0.0511880646925
FRO40251        SNA80324        1412    0.0454004694383
DAI75645        FRO40251        1254    0.0403202469374
FRO40251        GRO85051        1213    0.0390019613517
DAI62779        GRO73461        1139    0.0366226166361
DAI75645        SNA80324        1130    0.0363332368734
DAI62779        FRO40251        1070    0.0344040384554
DAI62779        SNA80324        923     0.0296775023311
DAI62779        DAI85309        918     0.0295167357963
ELE32164        GRO59710        911     0.0292916626475
DAI62779        DAI75645        882     0.0283592167454
FRO40251        GRO73461        882     0.0283592167454
DAI62779        ELE92920        877     0.0281984502106
FRO40251        FRO92469        835     0.026848011318
DAI62779        ELE32164        832     0.0267515513971
DAI75645        GRO73461        712     0.0228931545609
DAI43223        ELE32164        711     0.022861001254
DAI62779        GRO30386        709     0.02279669464
ELE17451        FRO40251        697     0.0224108549564
DAI85309        ELE99737        659     0.0211890292917
DAI62779        ELE26917        650     0.020899649529
GRO21487        GRO73461        631     0.0202887366966
DAI62779        SNA45677        604     0.0194205974084
ELE17451        SNA80324        597     0.0191955242597
DAI62779        GRO71621        595     0.0191312176457
DAI62779        SNA55762        593     0.0190669110318
DAI62779        DAI83733        586     0.018841837883
ELE17451        GRO73461        580     0.0186489180412
GRO73461        SNA80324        562     0.0180701585158
DAI62779        GRO59710        561     0.0180380052088
DAI62779        FRO80039        550     0.0176843188322
DAI75645        ELE17451        547     0.0175878589113
DAI62779        SNA93860        537     0.0172663258416
DAI55148        DAI62779        526     0.016912639465
DAI43223        GRO59710        512     0.0164624931674
ELE17451        ELE32164        511     0.0164303398605
DAI62779        SNA18336        506     0.0162695733256
ELE32164        GRO73461        486     0.0156265071863
DAI62779        FRO78087        482     0.0154978939584
DAI85309        ELE17451        482     0.0154978939584
DAI62779        GRO94758        479     0.0154014340375
DAI62779        GRO21487        471     0.0151442075817
GRO85051        SNA80324        471     0.0151442075817
ELE17451        GRO30386        468     0.0150477476608
FRO85978        SNA95666        463     0.014886981126
DAI62779        FRO19221        462     0.01485482781 9
DAI62779        GRO46854        461     0.0148226745121
```

```
DAI43223        DAI62779        459     0.0147583678981
ELE92920        SNA18336        455     0.0146297546703
DAI88079        FRO40251        446     0.0143403749076
cat: Unable to write to output stream.
```

HW 3.5 - Discussion of results

Again, we are running this job on a single Macbook Pro with 2 cores using two mappers and two reducers. This time though, **we find the job takes 1 minutes and 55 seconds to run**, with the mapper and reducer both getting called twice. This extra runtime is likely caused by the fact that reducing is a more computationally complex process when we're using stripes. Pairs, on the other hand, generates many more intermediate data points for the reducer to process, though their structure is simpler. The main benefit of the stripes paradigm is that it reduces the amount of network throughput required during a job, and it's in this exchange of data points that efficiencies can be created. However, since we're running this job on a single machine rather than a cluster, the network traffic savings don't outweight the additional computational complexity of the stripes reducer step.

We also explored whether or not we could improve performance by using a manual dictionary merge instead of the python Counter object, which carries some extra computational complexity. While this change helped slightly, it didn't make a major difference (and did increase the complexity of the code itself).

End of Submission