CS5540: Principles to Big Data - SP2017

Project 1 - Abdullah, Brendan, Noah and Sami

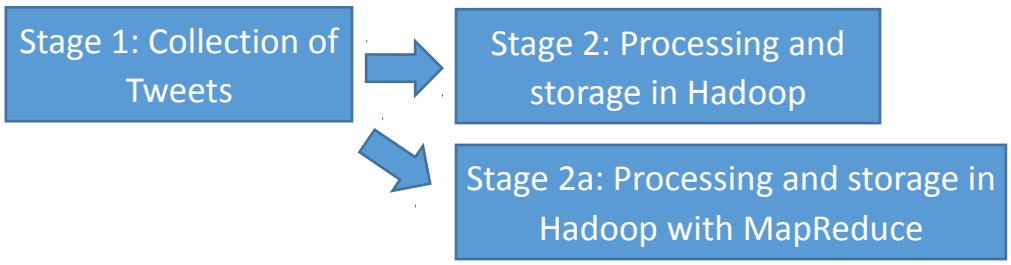
Project Overview:

The objective of the project is to collect 100,000 tweets from Twitter and identify the top 10 hashtags using the Hadoop file system. This report is split into 5 sections, as follows:

1. Collection of tweets from twitter with Twitter API
2. Processing of tweets to categorize by hashtags and identify top 10
3. Screenshots of Hadoop File System
4. Count Keywords (Extra Requirement)
5. Appendix: References, Source Code, Etc

The sections will include text, diagrams and source code where applicable to illustrate the methods and results of this project.

From a high level perspective, the algorithm flows as follows:



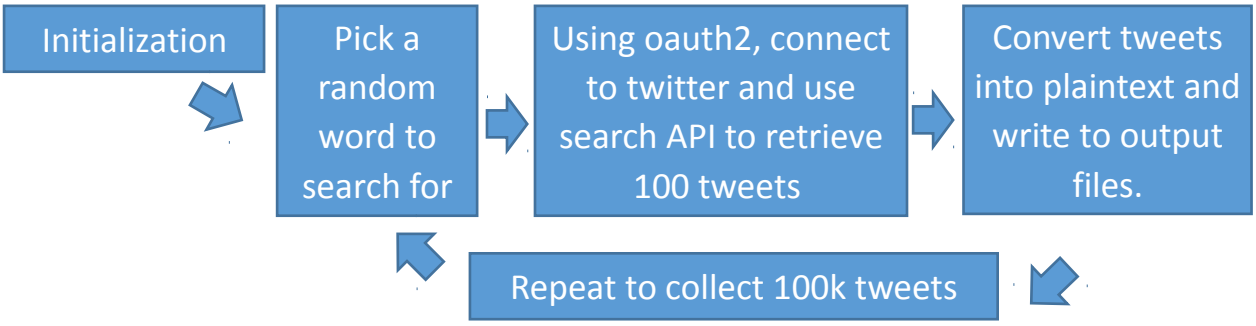
Section 1.

As dictated by the project requirements, 100K (100,000) tweets needed to be collected. To accomplish this Stage 1, this project uses the twitter API. By creating a twitter account and setting up an application, tweets can be collected via the Search API. For example, take this query:

https://api.twitter.com/1.1/search/tweets.json?q=the&count=100

This example would return 100 tweets containing the word "the". The tweets will be returned in a JSON format and will need to be processed into t for stage 2.

From a high level perspective, the flow diagram looks like so:



During initialization several libraries are imported, one of which is oauth2. In order to use certain twitter API queries, authentication must first be established. For this project, oauth2 was used; a function, oauth_req, is query twitter. The function will be explained later on.

```
def oauth_req(url, key, secret, http_method="GET", http_headers=None):
    consumer = oauth2.Consumer(key=CONSUMER_KEY, secret=CONSUMER_SECRET)
    token = oauth2.Token(key=key, secret=secret)
    client = oauth2.Client(consumer, token)
    resp, content = client.request( url, headers=http_headers)
    return content
```

Output files are created and prepared to be written to and a dictionary is opened and stored in a list to facilitate the picking or a random word. In addition, the consumer key and consumer secret are set.

A while loop is started, based on increment variable i. i will increment each time a tweet is written to an output file. A random word is chosen from the list of words that were initialized earlier.

```
# Loop to collect 100k tweets if possible (typically rate limited)
while i < 100000:
    randword = random.choice(word) # random word used to search twitter for tweets
    print str(i)
    print randword
```

With the random word chosen, the program will now query Twitter.

```
home_timeline = oauth_req('https://api.twitter.com/1.1/search/tweets.json?q='+ randword +'&count=100',
                          [redacted],
                          [redacted])
tweets = json.loads(home_timeline)  # load tweets into json parser
```

The function will return the tweets to home_timeline. Three arguments are passed into the oauth_req function.

The first is a string concatenated from three parts:

'https://api.twitter.com/1.1/search/tweets.json?q='+ randword +'&count=100'

If for example randword was chosen to be “bye”, the concatenated string would be:

'https://api.twitter.com/1.1/search/tweets.json?q=bye&count=100'

The twitter API will understand this to be a request for 100 tweets containing the text “bye”. The other two arguments, blacked out, are the token key and token secret. This program is meant to work with a Twitter App, and requires the key and secret for authentication. The earlier defined consumer secret and consumer token are also used for authentication.

home_timeline is then loaded into a JSON parser and stored in tweets. The tweets are converted to string type and then written to output files, one tweet per line. This process, random word, query and write, are repeated until 100,000 tweets are collected.

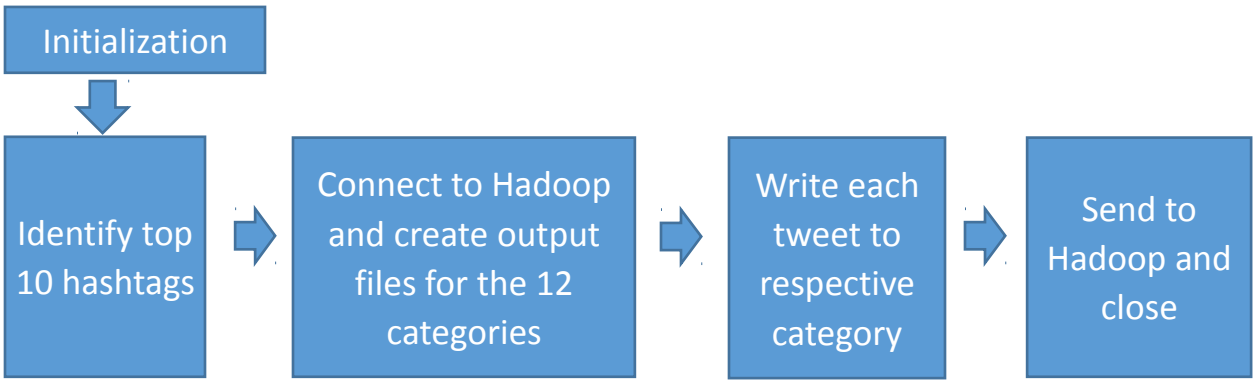
Write code snippet:

```
tweets = json.loads(home_timeline)  # load tweets into json parser

# Iterate through tweets and output each tweet to a file, 1 tweet per line
# Split output among 4 files to avoid one giant file
for t in tweets["statuses"]:
    tweet = str(t) + "\n"
    if i % 4 == 0:
        out1.write(tweet)
    elif i % 4 == 1:
        out2.write(tweet)
    elif i % 4 == 2:
        out3.write(tweet)
    elif i % 4 == 3:
        out4.write(tweet)
    i += 1
```

Section 2.

With the tweets collected, they must now be processed and sorted. Based on the top 10 most common hashtags, 10 categories will be created in the HDFS: one for each hashtag. Two additional categories will also be created to store tweets that either have no hashtags or have a less common hashtag.

From a high level perspective, the flow diagram looks like so:



Initialization will import relevant libraries, such as hdfs which is needed to interface with Hadoop.

```
import ast
import os
import hdfs
import operator


# Helper function to identify which hashtag category a tweet belongs to
def tagCategory(hashtags, word):
    for tup in hashtags:
        tag = tup[0]
        if tag == word:
            return word
    return "OTHER"

# Open tweet input file
fin = open("allout.txt", "r")
tweets = fin.readlines()

hashtagdict = {}
```

The function tagCategory will be explained later. The file containing the tweets is opened and stored in a list. Finally a dictionary is initialized. This dictionary, hashtagdict, will store a count of hashtag occurrences to identify the top 10.

Using a for-loop, each tweet's 'hashtags' property is stored in ht. Since a tweet can have multiple hashtags, a for-loop is used to check each one. If the hashtag is already in the dictionary, simply increment the value. If there is no entry for that hashtag, then set it to one.

After going through every hashtag, in every tweet, the dictionary's items are then sorted so that the entries with the most occurrences are placed at the front. Store those first ten into a list called topten.

```
i = 0
for line in tweets:
    # t = json.loads(ast.literal_eval(line))
    tweet = ast.literal_eval(line)["text"]
    ht = ast.literal_eval(line)["entities"]['hashtags']
    print (str(i) + " : " + tweet)
    if (ht != []):
        h = 1
        for tag in ht:
            print ("hashtag #" + str(h) + " " + tag["text"])
            if (hashtagdict.has_key(tag["text"])):
                hashtagdict[tag["text"]] += 1
            else:
                hashtagdict[tag["text"]] = 1
            h += 1
    i += 1

hashtagsort = sorted(hashtagdict.items(), key=operator.itemgetter(1), reverse=True)
topten = hashtagsort[:10]
```

With a list of the top ten hashtags, the tweets must now be placed into the Hadoop File system. This is accomplished using the hdfs library.

First a connection is established to the HDFS server, which is run locally. Using the topten list, ten directories are created in the HDFS system; a map from the tags to the directories are also created. Two additional directories are set to complete the

12 categories. Output files are then set to store the tweets. For categories of "OTHER" and "NONE", the tags are set directly to the text file.

```
# Create connection to hadoop and create map from tags to their respective directory in hadoop
tagToHadoopDir = {}
client = hdfs.InsecureClient("http://localhost:50070", user="hduser")
for tup in topten:
    path = "/user/hduser/" + tup[0]
    client.makedirs(path)
    tagToHadoopDir[tup[0]] = path
client.makedirs("/user/hduser/Others")
client.makedirs("/user/hduser/None")

# Create a directory for convince to store category files
if not os.path.exists("tweetfiles"):
    os.makedirs("tweetfiles")

# Create output files for each tweet category
tagToOutFile = {}
for tup in topten:
    strg = "tweetfiles/"+tup[0]+"-tweets.txt"
    tagToOutFile[tup[0]] = open(strg,"w")
tagToOutFile["OTHER"] = open("tweetfiles/Other-tweets.txt","w")
tagToOutFile["NONE"] = open("tweetfiles/None-tweets.txt", "w")
```

The program now begins writing tweets to the appropriate categories. Similar to before, the tweet's hashtags are stored in ht. If ht is null (no hashtag) then write to the file tagged as NONE. This is accomplished using the tagToOutFile function. Otherwise, check each hashtag and write to the appropriate file.

```
# Write each tweet to their respective tweet category
# options are one of the top ten, other, or none
for line in tweets:
    ht = ast.literal_eval(line)["entities"]["hashtags"]
    if ht == []:
        tagToOutFile["NONE"].write(line)
    for tag in ht:
        tagcat = tagCategory(topten,tag["text"])
        tagToOutFile[tagcat].write(line)
```

A key function is the tagCategory function, which was defined earlier. If the tweet's hashtag is among the top ten, it will be returned to be used as a tag. If the tweet's hashtag is not among the top ten, "OTHER" will be returned to be used as a tag.

```
# Helper function to identify which hashtag category a tweet belongs to
def tagCategory(hashtags, word):
    for tup in hashtags:
        tag = tup[0]
        if tag == word:
            return word
    return "OTHER"
```

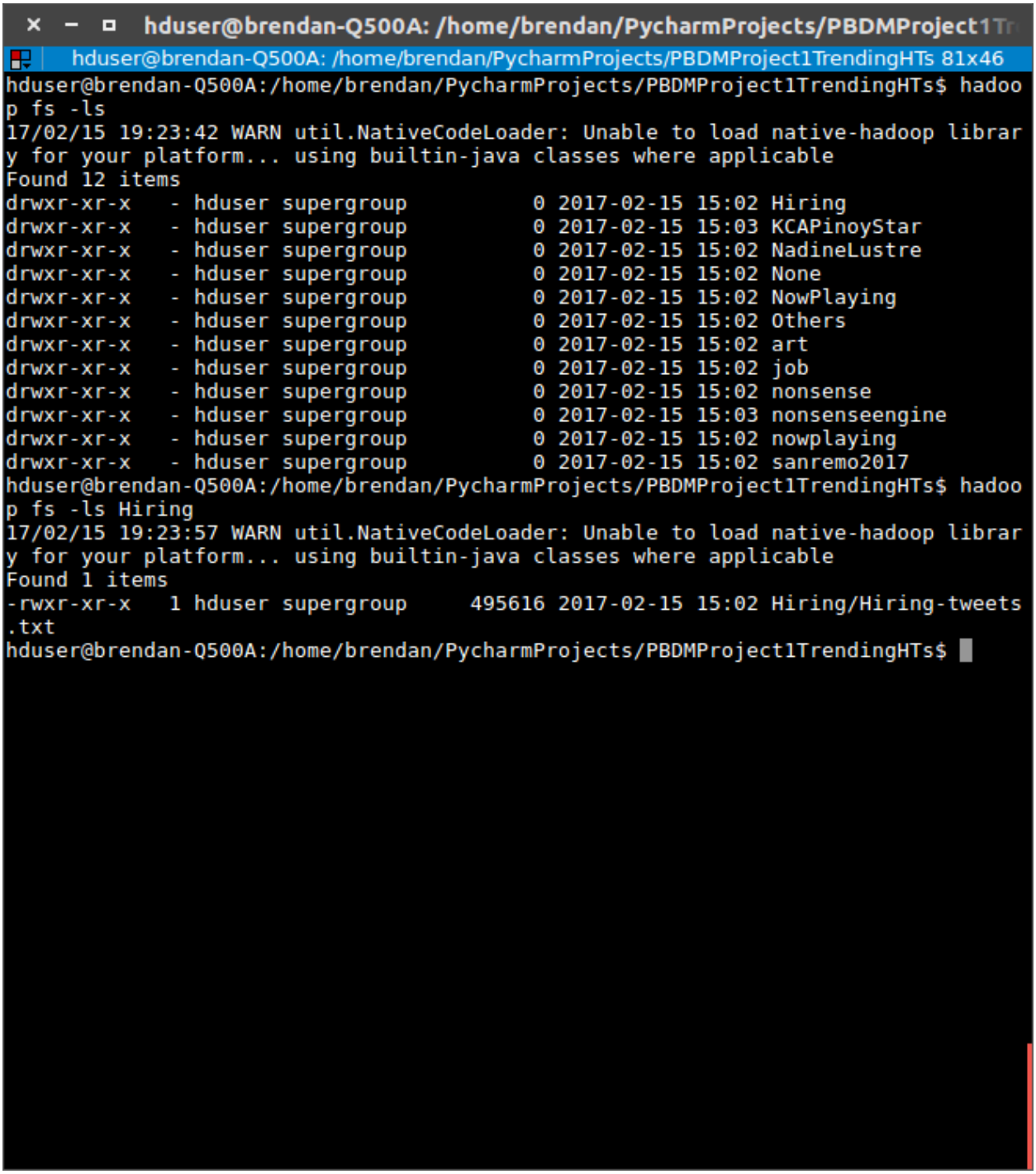
After all tweets have been written to the appropriate files, the program will tell Hadoop to close the files and finish up.

```
# Send to hadoop and close each file
for tag, file in tagToOutFile.iteritems():
    if tag == "OTHER":
        hdir = "/user/hduser/Others"
    elif tag == "NONE":
        hdir = "/user/hduser/None"
    else:
        hdir = tagToHadoopDir[tag]
    client.upload(hdir, file.name)
    file.close()

print (client.list("/user/hduser/"))
fin.close() # close tweets input file
```

With the program complete, the project's main requirement has been fulfilled.

Section 3.

Screenshots of the HDFS after main program completion.



Tweets with no hashtags:

Tweets from “OTHER” category:

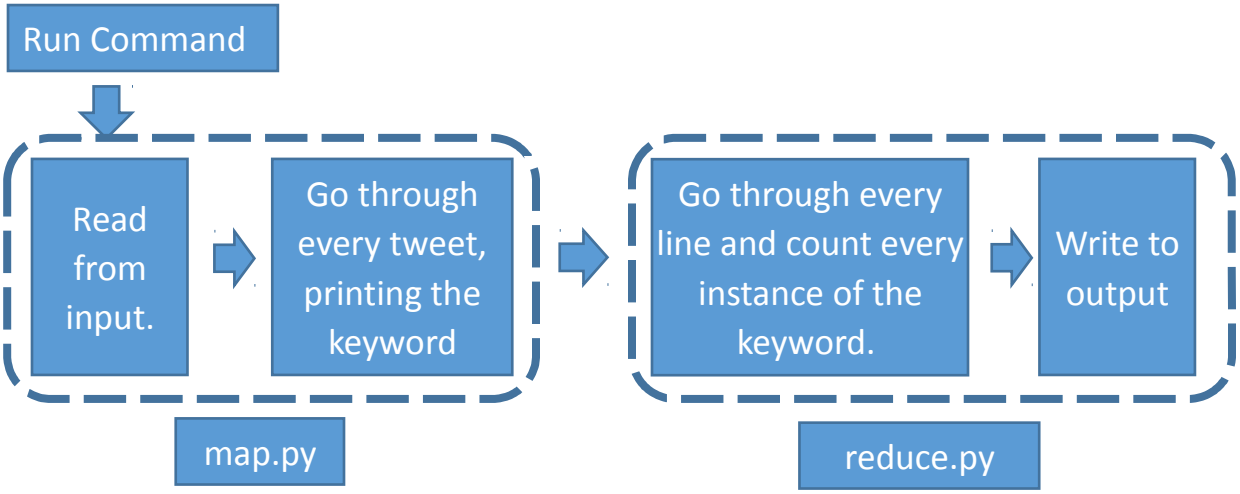
Section 4.

An extra requirement of the project is to implement a program that counts the number of times a keyword appears in either the tweet's text or hashtag. For this project, a MapReduce algorithm was used.

```
hduser@brendan-Q500A: /home/brendan/PycharmProjects/PBDMProject1Tr
hduser@brendan-Q500A: /home/brendan/PycharmProjects/PBDMProject1TrendingHTs 81x46
hduser@brendan-Q500A:/home/brendan/PycharmProjects/PBDMProject1TrendingHTs$ hadoo
p jar hadoop-streaming-2.7.3.jar -mapper map.py -reducer reduce.py -input /user/h
duser/* -output Output
```

The MapReduce code is split across two files: one for the mapper and the other for the reducer. The input will pull from the 12 directories created earlier. For this project, the keyword "Amazon" was used.

From a high level perspective, the flow diagram looks like so:



For the mapper, it uses a for-loop is used to read in the tweets from the input directories.

```
import sys, ast

keyword = "Amazon"
attribute = "hashtags"

for line in sys.stdin:
```

For each line, two attributes are checked – “text” and “hashtags”. Both are checked for he keyword. Both text and hashtag portions are split by word and checked against the keyword. If the word matches, it is printed.

```
if attribute == "text":
    try:
        newline = ast.literal_eval(line)["text"]
        newline = newline.strip()
        words = newline.split()
        for word in words:
            if word == keyword:
                print '%s\t%s' % (word, 1)
    except:
        continue


elif attribute == "hashtags":
    try:
        hts = ast.literal_eval(line)["entities"]["hashtags"]
        if not hts == []:
            for ht in hts:
                if ht["text"] == keyword:
                    print '%s\t%s' % (ht["text"], 1)
    except:
        continue
```

For the reducer, it uses a for-loop to read from the prints made by the mapper.

```
current_word = None
current_count = 0
word = None

for line in sys.stdin:
```

Three variables are set: current_word and current_count will hold the final count, and will be printed to output.

For each line, it is processed so that the word (“Amazon”) and count (“1”) are placed into variables. count is also converted into an int.

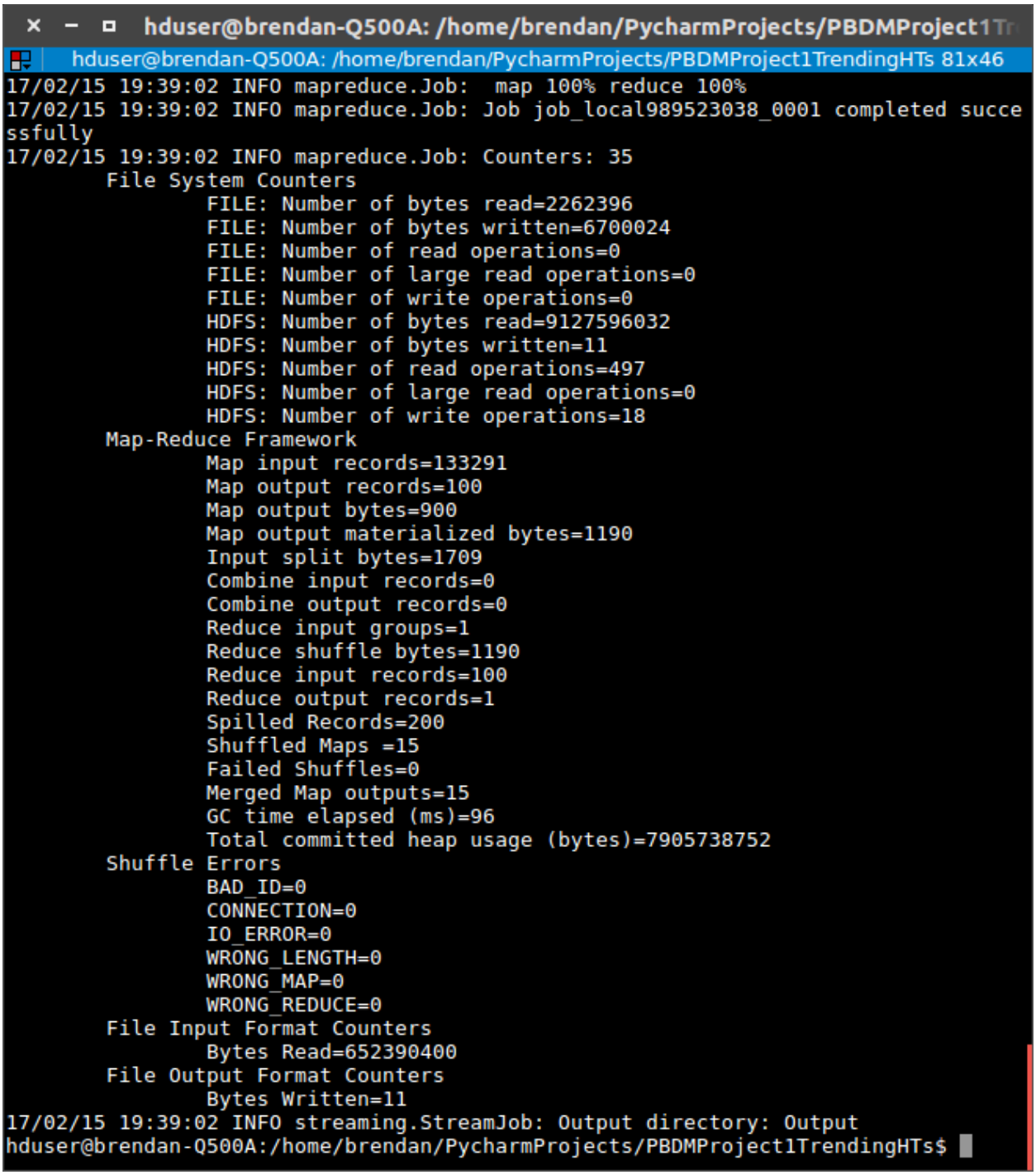
```
line = line.strip()

word, count = line.split('\t', 1)

try:
    count = int(count)
except ValueError:
    continue
```

In brief, the counting portion is taken care of by an if/else. Since current_word is initially set to none, the else conditional will trigger. This will set the current_count and current_word to the inputted line. From there, future iterations should trigger the if conditional, updating the current_count. Once all lines have been read in, a final print will be made. The program is complete and has counted every keyword appearance in the tweet’s text and hashtag.

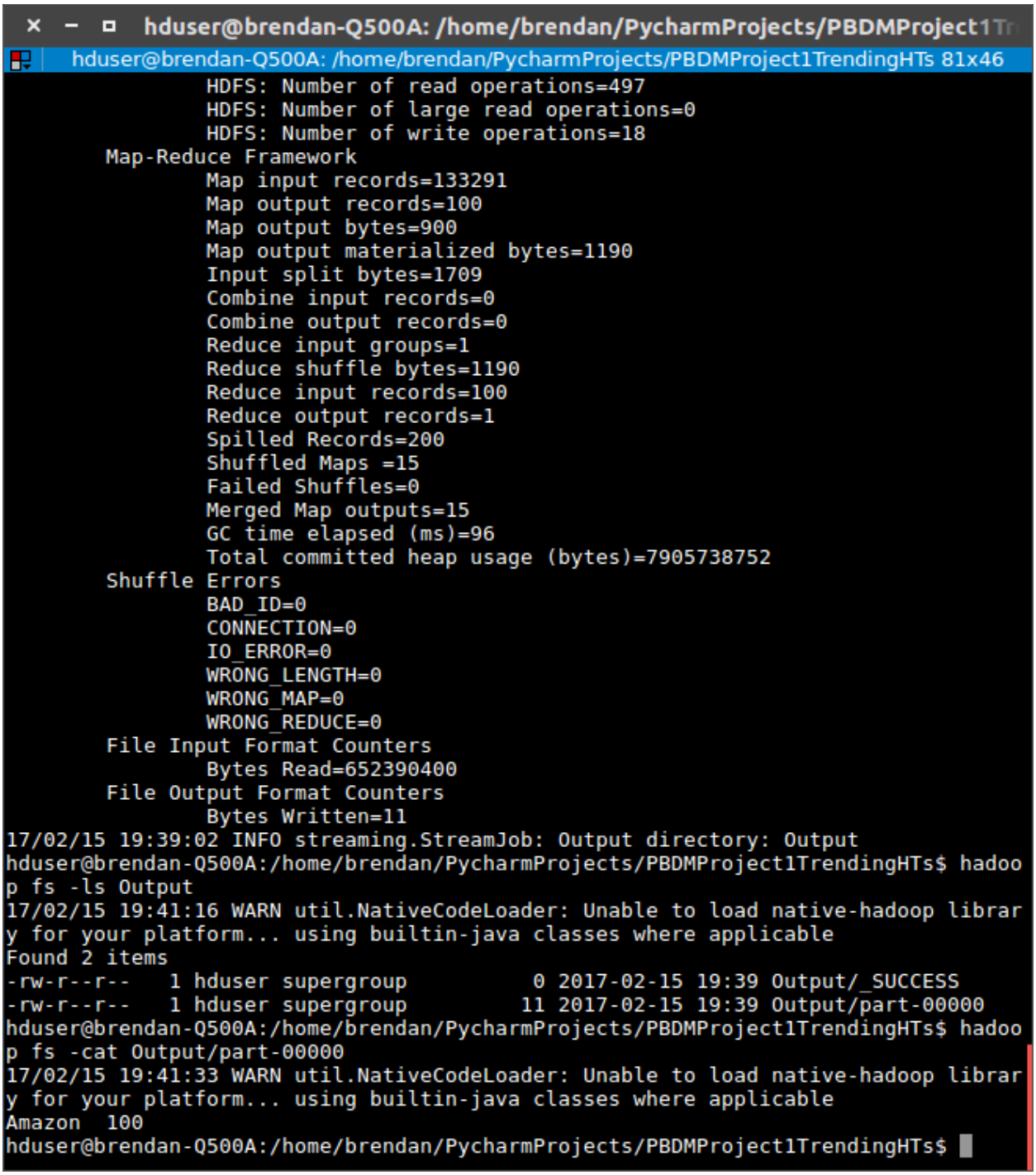
```
    if current_word == word:
        current_count += count
    else:
        if current_word:
            print '%s\t%s' % (current_word, current_count)
        current_count = count
        current_word = word

if current_word == word:
    print '%s\t%s' % (current_word, current_count)
```

Below are two screenshots of the console after the command was executed. At the bottom of the second screenshot, the final result can be seen; it is indicated by a blue arrow.

First Screenshot



Second screenshot – see blue arrow for final result



Section 5.

References:

- Twitter API:
  - https://dev.twitter.com/docs
- Twitter Search API:
  - https://dev.twitter.com/rest/public/search
- Twitter Authentication API:
  - https://dev.twitter.com/oauth/application-only
- OAuth2 Python Library:
  - https://github.com/joestump/python-oauth2
- OAuth example:
- https://dev.twitter.com/oauth/overview/single-user
- HDFS Install:
  - http://www.bogotobogo.com/Hadoop/BigData_hadoop_Install_on_ubuntu_single_node_cluster.php
- HDFS Quickstart:
  - https://hdfscli.readthedocs.io/en/latest/quickstart.html#configuration
- Check if Directory Exists:
  - http://stackoverflow.com/questions/273192/how-to-check-if-a-directory-exists-and-create-it-if-necessary
- Sorting Dictionary:
  - http://stackoverflow.com/questions/613183/sort-a-python-dictionary-by-v`alue
- Hadoop MapReduce
  - https://hadoop.apache.org/docs/r2.6.0/hadoop-mapreduce-client/hadoop-mapreduce-client-core/HadoopStreaming.html
- Python/Hadoop MapReduce
  - http://www.michael-noll.com/tutorials/writing-an-hadoop-mapreduce-program-in-python/

Source Code: gettweets.py, gettophts.py, map.py, reduce.py

gettweets.py:

import oauth2

```
import json
import random

# Twitter provided access key and secret
CONSUMER_KEY = "[REMOVED]"
CONSUMER_SECRET = "[REMOVED]"

# Twitter provided function to submit authenticated requests for data
def oauth_req(url, key, secret, http_method="GET", http_headers=None):
   consumer = oauth2.Consumer(key=CONSUMER_KEY, secret=CONSUMER_SECRET)
   token = oauth2.Token(key=key, secret=secret)
   client = oauth2.Client(consumer, token)
   resp, content = client.request( url, headers=http_headers)
   return content

out1 = open("output1.txt", "a") #output files
out2 = open("output2.txt", "a")
out3 = open("output3.txt", "a")
out4 = open("output4.txt", "a")
words = open("/usr/share/dict/words", "r") #dict of words to random search for
word = words.readlines()
i = 0

# Loop to collect 100k tweets if possible (typically rate limited)
while i < 100000:
   randword = random.choice(word) # random word used to search twitter for tweets
   print str(i)
   print randword

   home_timeline = oauth_req('https://api.twitter.com/1.1/search/tweets.json?q='+ randword
+'&count=100',
                     '[REMOVED]',
                     '[REMOVED]')
   tweets = json.loads(home_timeline)  # load tweets into json parser

   # Iterate through tweets and output each tweet to a file, 1 tweet per line
   # Split output among 4 files to avoid one giant file
```

```
    for t in tweets["statuses"]:
        tweet = str(t) + "\n"
        if i % 4 == 0:
            out1.write(tweet)
        elif i % 4 == 1:
            out2.write(tweet)
        elif i % 4 == 2:
            out3.write(tweet)
        elif i % 4 == 3:
            out4.write(tweet)
        i += 1

out1.close()
out2.close()
out3.close()
out4.close()
words.close()
```

gettophts.py:

```
import ast
import os
import hdfs
import operator


# Helper function to identify which hashtag category a tweet belongs to
def tagCategory(hashtags, word):
    for tup in hashtags:
        tag = tup[0]
        if tag == word:
            return word
    return "OTHER"

# Open tweet input file
fin = open("allout.txt", "r")
tweets = fin.readlines()

hashtagdict = {}

i = 0
for line in tweets:
    # t = json.loads(ast.literal_eval(line))
    tweet = ast.literal_eval(line)["text"]
    ht = ast.literal_eval(line)["entities"]['hashtags']
```

```
    print (str(i) + " : " + tweet)
    if (ht != []):
        h = 1
        for tag in ht:
            print ("hashtag #" + str(h) + " " + tag["text"])
            if (hashtagdict.has_key(tag["text"])):
                hashtagdict[tag["text"]] += 1
            else:
                hashtagdict[tag["text"]] = 1
            h += 1
    i += 1

hashtagsort = sorted(hashtagdict.items(), key=operator.itemgetter(1), reverse=True)
topten = hashtagsort[:10]

# topten = [(u'nonsenseengine', 530), (u'nonsense', 530), (u'KCAPinoyStar', 280),
#       (u'NadineLustre', 219), (u'job', 217), (u'art', 154), (u'NowPlaying', 136),
#       (u'Hiring', 129), (u'sanremo2017', 126), (u'nowplaying', 120)]

# Create connection to hadoop and create map from tags to their respective directory in hadoop
tagToHadoopDir = {}
client = hdfs.InsecureClient("http://localhost:50070", user="hduser")
for tup in topten:
    path = "/user/hduser/" + tup[0]
    client.makedirs(path)
    tagToHadoopDir[tup[0]] = path
client.makedirs("/user/hduser/Others")
client.makedirs("/user/hduser/None")

# Create a directory for convince to store category files
if not os.path.exists("tweetfiles"):
    os.makedirs("tweetfiles")

# Create output files for each tweet category
tagToOutFile = {}
for tup in topten:
    strg = "tweetfiles/"+tup[0]+"-tweets.txt"
    tagToOutFile[tup[0]] = open(strg,"w")
tagToOutFile["OTHER"] = open("tweetfiles/Other-tweets.txt","w")
tagToOutFile["NONE"] = open("tweetfiles/None-tweets.txt", "w")

# Write each tweet to their respective tweet category
# options are one of the top ten, other, or none
for line in tweets:
```

```
    ht = ast.literal_eval(line)["entities"]["hashtags"]
    if ht == []:
        tagToOutFile["NONE"].write(line)
    for tag in ht:
        tagcat = tagCategory(topten,tag["text"])
        tagToOutFile[tagcat].write(line)

# Send to hadoop and close each file
for tag, file in tagToOutFile.iteritems():
    if tag == "OTHER":
        hdir = "/user/hduser/Others"
    elif tag == "NONE":
        hdir = "/user/hduser/None"
    else:
        hdir = tagToHadoopDir[tag]
    client.upload(hdir, file.name)
    file.close()

print (client.list("/user/hduser/"))
fin.close() # close tweets input file
```

map.py:

```
#!/usr/bin/python
import sys, ast

keyword = "Amazon"
attribute = "hashtags"

for line in sys.stdin:
    if line == "":
        continue

    if attribute == "text":
        try:
            newline = ast.literal_eval(line)["text"]
            newline = newline.strip()
            words = newline.split()
            for word in words:
                if word == keyword:
                    print '%s\t%s' % (word, 1)
        except:
            continue
```

```
    elif attribute == "hashtags":
        try:
            hts = ast.literal_eval(line)["entities"]["hashtags"]
            if not hts == []:
                for ht in hts:
                    if ht["text"] == keyword:
                        print '%s\t%s' % (ht["text"], 1)
        except:
            continue
```

reduce.py:

```
#!/usr/bin/python
from operator import itemgetter
import sys

current_word = None
current_count = 0
word = None

for line in sys.stdin:
    line = line.strip()

    word, count = line.split('\t', 1)

    try:
        count = int(count)
    except ValueError:
        continue

    if current_word == word:
        current_count += count
    else:
        if current_word:
            print '%s\t%s' % (current_word, current_count)
        current_count = count
        current_word = word

if current_word == word:
    print '%s\t%s' % (current_word, current_count)
```