Alex Smith
W205

ASSIGNMENT #4

OVERVIEW

I wrote fifawwc.py as the main python program for this assignment. This program calls upon tweetscrape.py to generate the Twitter URLs that I wish to scrape. These URLs print to the console for verification by the user. The user then copies and pastes the URLs in the start_urls field of tweetspider.py. tweetspider.py scrapes all the URLs provided to it in the start_urls field. fifawwc.py will also call archiveindex.py which takes the CSV file generated by the spider and indexes it for easy querying.

mwordcount.py is the mapper and rwordcount.py is the reducer for the MapReduce function to count all words that occur more than 10,000 times. It is important to turn both these programs into executable programs using the “sudo chmod +x” command. The user can run the MapReduce function with the command line command: “./mwordcount.py | sort | ./rwordcount.py”

ARCHITECTURE & DESIGN

In tweetscrape.py, I choose to create a different start URL for each date. This ensures that I get a representative sample of tweets across the specified date range. Because Twitter sorts the tweets by date when searching, had I not made this decision, I would have only ended up with a small number of dates unless I scraped every tweet.

In tweetspider.py, I choose to concatenate all the hashtags into a single string separated by spaces. A single hashtag never includes a space character. This design decision allows me to keep all my hashtags in a single field in the CSV file. When analyzing this field, I can always do a simple split() method to divide the concatenated hashtag field into its component words.

In tweetspider.py, I choose to add each tweet item to the CSV as it is scraped. Scrapy spiders write over their “items” files as they move between start_urls. This ensures that each tweet is properly stored to the CSV before the spider begins crawling the next website.

I did not have time to finish this assignment and complete the remaining MapReduce programs.

RESILIENCY

One of the key parts of my program’s resiliency lies in the intentional modes with which files are read from and written to. When writing the first tweet to the CSV file, the tweetspider.py first opens the file in write mode. This allows the program to create the file and write the first tweet. The program then closes the file. When writing all subsequent tweets, the tweetspider.py opens the CSV file in append mode. By opening the file in append mode, the program protects the previous tweets written to the file.

In the archiveindex.py file, I added an “if” statement to help the program skip over blank lines in the CSV file. The program treats each line as a list of 6 elements. However, blank lines only have one element. The “if” statement ensures that the program does not fail on these blank lines.