Practical 05   CS1003   120013647   April 29th, 2014

TwitSubStringMapper- This method searches for a substring within the text of a tweet. It does this by parsing the JSON file for the text and then using the contains method to see if the string of characters is within the text

Reducers- the reducer classes only vary in name and were copied directly from the lecture example slides. I had a difficult time knowing exactly what the reducer class was supposed to do over the mapper class. In every instance, I found it easier to simply moderate the input into the mapper rather than trying to edit it in the reducer.

TwitterDriver- This is the main class for the set of classes that find the substrings. As with the reducer, much of this is copied from the lecture slides. The main difference are how the user now has to type in exactly where they want the files to be read from and where they want it to end up. The JobClient.runJob(job) method is commented out because I would only ever run one class at a time. It must be uncommented in order to run. Originally, I tried to pass a parameter to the mapper methods that the user would input to specify dates or a substring to search, but the I had a lot of difficulty implementing the Context class and could not get it to work.

TwitTimeMapper-This method prints out all tweets between two given time periods. The static strings in the beginning of the method are the two dates. These can be changed all will. This functions very similarly to the other mapper, but it invokes two methods to convert the JSON string to a usable date and then another to compare them.

TwitTimeDriver-This method is identical to the other driver method.

MostPopularUserMapper- This method simply adds the usernames of users mentioned in tweets to the map. It does this by looking for the @ symbol which is at the beginning of every username. The string is cleaned first though before being added to the map. The CleanWord method was copied from the Zipf's law practical.

MostPopularUserDriver-The main difference in this class is the cluster of methods at the bottom that have been commented out. They are commented out so that other methods can run and must be uncommented to run the driver.

Top3-I researched how to reorder a map based on values and learned this must be done through a secondary sort. I could not figure out how to actually implement this technique though because I do not entirely understand several of Hadoop's built in methods and types. Instead, I recycled significant chunks of the Zipf's law practical and created an admittedly crude and roundabout solution.

The MostPopularUserReducer creates a file with all of the users that have been tweeted and the number of times. Top3 then reads the file in, splits them into a an array, and then adds them to another HashMap. This new HashMap is then sorted by ascending value. Using a series of for loops, I inverted the HashMap and then call the keys to get the value for the three most popular tweeters. These are then written to an output file.

General difficulties- Overall, I found it difficult to implement Hadoop because it seemed at times like normal Java I/O techniques would work. This would not be the case for a larger data set or if the program was actually distributed over several computers. I especially found it difficult to know what the reducer function should do because manipulating the map after everything was already entered into it was very cumbersome. Normally, consulted the online API is very helpful, but since Hadoop updated their API recently many of the web sources were out of data which made researching solutions much more difficult.

Improvements from practical P04-I improved on last time by making the user specify where they want the file to be read from as opposed to hard coding it in. I also used more classes and less static methods because making too many static methods defeats the point of object oriented programming.