Nicholas Achin
COMPSCI 446
Indexing Report

1. For my implementation of indexing, I followed predominantly what was taught in class. I created the index by nesting a SortedMap inside of a HashMap. The structure was HashMap< terms, <SortedMap<DocId, Posting>>. This allowed me to keep track of all the documents a term occurred in and the positions in the documents. I created a Posting object to store the positions, playId, and to access these when performing queries/updating the index. Once I created my index, I created a few query helper functions. The booleanQuery function I created can check for the existence of an array of strings in the index. This method is quite robust and would do well on larger term-occurrence based queries. In this method, I used a boolean flag to differentiate when I wanted to return plays versus scenes. This is where storing the playId in the Posting object was very useful. Another query method I created was for counting occurrences of documents where thee and thou occurred more than you. This method is hardcoded for this specific query, but I did this out of efficiency. I could have written a more robust method that looped through each of the arrays of queries and compared each term one by one. I found it more efficient to add all the occurrences of "thee" and "thou" to my set and then loop through the posting list of "you" and remove documents where "you" occurred more often than or equal to "thee" and "thou". My final helper METHOD

   I wrote another helper method to handle the output of the sets. I passed it the set and output file and it looped through and printed each play/scene line by line in alphabetical order. Another helper function I had was to compute the statistics needed for the report. I kept track of the longest and shortest plays as I read in each play from the iterator. I sent that data along with a HashMap of all of the scenes to my helper function where I found the shortest scene in the HashMap using a comparator and the average scene length by using simple math. I outputted that to stats.txt.
2. Software Libraries Used
   a. Org.json.simple.*;
      i. I used json-simple's library as it was suggested for parsing JSON for the index. I read in the JSON as a JSON object and then converted the "corpus" array into a JSON array and looped through that to index sceneId's and the posting lists from the text terms.
   b. Java.io.*;
      i. I needed to use java.io to read and write files as well as handle any input/output errors that may occur in these processes.
   c. Java.util.*;
      i. I used Java's util library for many different aspects of this project. I need the HashMap to create the index and the SortedMap to nest the sorted map inside of the index. I also used Sets for outputting the data in alphabetized fashion. I used ArrayLists for storing the positions of terms

in my Posting objects and used it to store a phrase queries SortedMaps for each query term.

3. Counts could be misleading because the length of scenes is variable. You could fix this by normalizing the data by creating a count per scene length (count / scene word count) score. This would show the scenes where the word makes up a bigger percentage of the word usage.
4. The phrase based queries take much longer because they require looping through the index to find all the scenes that contain all the query words. Then they require more looping to compare all of the posting lists of these terms.
5. The shortest scene was antony_and_cleopatra:2.8 at 47 words. The average scene length was 1199.556 words. The shortest play was comedy_of_errors at 16,415 words. The longest play was hamlet at 32,867 words.
6. To be determined…