Winston: Shakespeare Query Web Portal
CS560: Programming Assignment 2

Elliot D. Greenlee and Jared M. Smith

March 31st, 2017

1 Introduction

We implemented a web portal capable of querying the works of Shakespeare as they appear in the First Folio for the second programming assignment in CS560: Advanced Operating Systems. We implemented the text analysis components of the project in Java and the web portal in Python. The text analysis portion was run on a Hadoop cluster on Cloudlab using the MapReduce programming model. We named our system after Winston, a character from Overwatch (`https://playoverwatch.com/en-us/`), a modern team-oriented combat game, because this particular character is resilient and highly intelligent, quoting Shakespeare to inspire his teammates. The ultimate goal of our system is to provide an easy to use resource for others to explore Shakespeare's plays.

Figure 1: Winston: An Overwatch Hero

2 Usage

Before running Winston, you have to generate the stopwords and inverted index using Hadoop, either locally or on Cloudlab. We have already generated the inverted_index.txt file, which is the inverted index on all of Shakespeare's plays, and is stored in the Web UI folder and in the Part 2 code folder.

The Winston Web UI is run locally with Python 2 by executing the command:

```
pip install -r requirements.txt
```

to bootstrap your environment and

```
honcho -f Procfile start
```

to start the user interface. At this point, the portal can be accessed using your browser by navigating to **localhost:5000**.

3 Implementation

We implemented Winston as two separate portions in Java and Python. The internal generation and text analysis is done using Java on a Hadoop Cloudlab cluster. We used the MapReduce model for this text analysis. First, the documents are preprocessed to remove capitalization and stop words using a word frequency cutoff. Next, an inverted index of all words is generated mapping each to a list of its locations by document, line, and word position. The external user interface is written as a Flask app in Python. It allows user queries on the pre-generated inverted index file, returning the locations in a separate window.

3.1 Stop Words

Where is the Code: The code for the stopwords MapReduce tasks is in part1_stopwords/StopWord.java.

During text analysis, there are "stop words" which are so common that they only provide noise instead of interesting information in a document. Typically these stop words are identified and removed as a preprocessing step. We chose to perform this function using a word count MapReduce function on the Hadoop Cloudlab cluster. In total, we removed 119 different words from the document based on a frequency threshold, with our least frequent removal being the word "take" at 4047 occurrences. This decision was made based on eyeballing from most frequent to least frequent words, and deciding when interesting words began to appear. Additionally, we determined our number of stop words is similar to the count in a number of well curated lists. Words were all converted to lowercase before analysis. The stop list and word counts are available in the source code provided, in part1_stopwords/stopwords_output.csv.

3.2 Inverted Index

Where is the Code: The MapReduce code for building the inverted index is in part2_inverted_index_builder/InvertedIndexBuilder.java.

An inverted index stores mappings from some content to its location in a larger database. In this assignment, we created an inverted index mapping words to their locations in Shakespeare's First Folio. This location data consisted of a document ID, line number, and line index, all of which are needed for the later querying ability. This functionality was performed using a MapReduce function on the Hadoop Cloudlab cluster.

3.3 Query

Where is the Code: The Python code for querying the inverted index is in part3_inverted_index_query/query_inverted_index.py.

The final goal of the project was a querying program on top of the full inverted index. It is a Python 2 program which accepts a user-specified query of one or more words and returns the relevant location document IDs, line numbers, and line indexes. Currently, our querying functionality is limited to uncomplicated multiple word queries. However, common operations like "and," "or," and "not" would be feasible on our system. Additionally, we have implemented the underlying word-position logic to search for consecutive words using quote specification such as in the example "fare well." In the future this query would only return the lines where these two words appear consecutively.

3.4 Web Portal

Where is the Code: The MapReduce code for building the inverted index is in part4_extra_credit_web_ui/searchui.

We further improved the query by implementing it in a Python 2 web portal, similar to a real search engine. It allows the submission of queries, and then displays the results in a returned page. We built the UI as a Flask App (`https://github.com/sloria/cookiecutter-flask`).

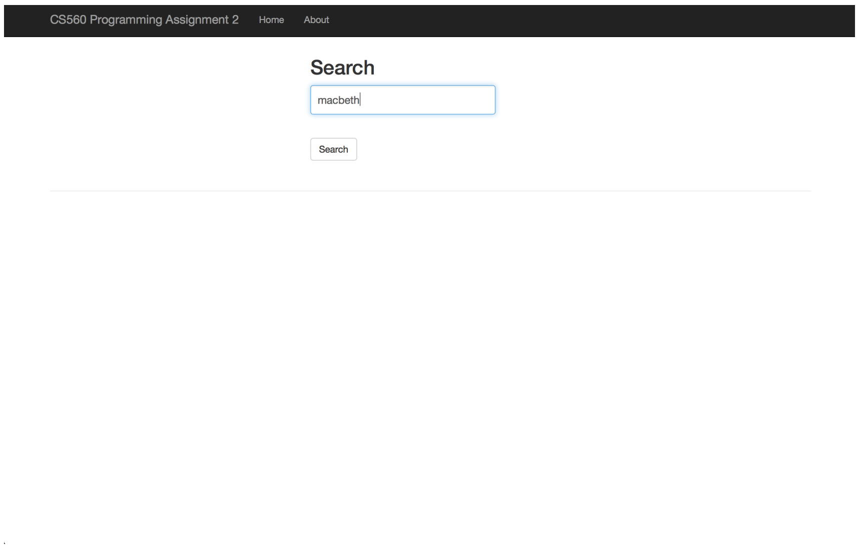
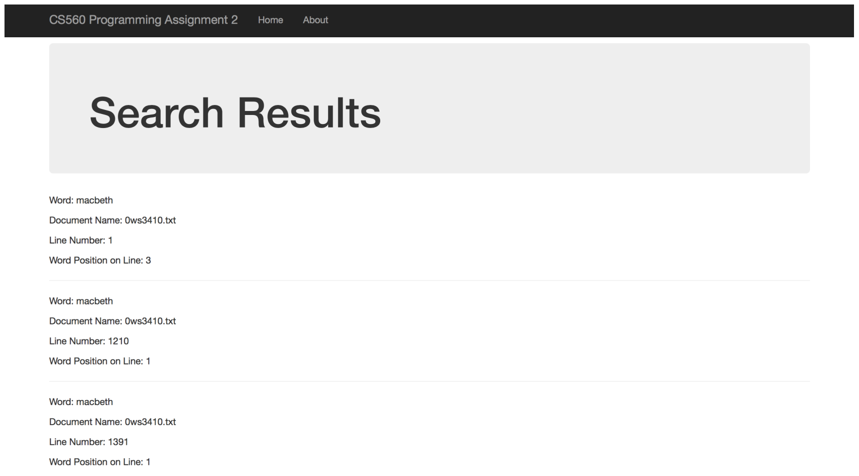


Figure 2: The web portal initial search page



Search Results

Word: macbeth
Document Name: 0ws3410.txt
Line Number: 1
Word Position on Line: 3

Word: macbeth
Document Name: 0ws3410.txt
Line Number: 1210
Word Position on Line: 1

Word: macbeth
Document Name: 0ws3410.txt
Line Number: 1391
Word Position on Line: 1

Figure 3: The web portal results page

4 Test Results

We ran a few basic queries through our web portal to show functionality. Here are those results.

First, we search for (mercutio), a character in Romeo and Juliet. We can see that all the document names are the same.

```
Word: mercutio
Document Name: 0ws1610.txt
Line Number: 1344
Word Position on Line: 3

Document Name: 0ws1610.txt
Line Number: 1352
Word Position on Line: 2

Document Name: 0ws1610.txt
Line Number: 1779
Word Position on Line: 3

Document Name: 0ws1610.txt
Line Number: 2315
Word Position on Line: 3

Document Name: 0ws1610.txt
Line Number: 2394
Word Position on Line: 8

Document Name: 0ws1610.txt
Line Number: 2406
Word Position on Line: 5

Document Name: 0ws1610.txt
Line Number: 2447
Word Position on Line: 7

Document Name: 0ws1610.txt
Line Number: 798
Word Position on Line: 6
```

Next, we search for two words that are attributed to Shakespeare, "assassination" and "fashionable" in the query (fashionable assassination).

```
Word: fashionable
Document Name: 0ws3710.txt
Line Number: 3271
Word Position on Line: 6
```

Word: assassination
Document Name: 0ws3410.txt
Line Number: 993
Word Position on Line: 6

5 Challenges

We faced several challenges while building Winston.

- The obvious challenge of this project was working under a new environment and model than used in typical solutions. Programs are run distributed automatically by Hadoop and Cloudlab across the cluster. Additionally, rather than common function calls and data passing, the MapReduce model is used. This obfuscates some of the details, and requires some trust in the system, meaning that debugging is usually slower and cannot rely on tracing.

- We also ran into issues where our inverted index was being built incorrectly. When using Hadoop, hadoop does not track the line numbers of input files within the mapper tasks, so we had to write a program under the **utils** folder to add line numbers to any set of input. This input was used in the mapper task to keep track of the line numbers and further drill down on the exact word position of each word in the lien.

6 Conclusion

In summary, we built Winston, a query web portal capable of querying the works of Shakespeare. To support this web portal, we implemented text analysis tools that run on a Hadoop and Cloudlab cluster. This text analysis was conducted over two steps: a pre-processing step to remove stop words, and an inverted index calculator. This inverted index was then used to allow for queries that return word location by document, line, and word position in the works of Shakespeare. Our implementation was successful, allowing for interesting queries about where character names appear throughout certain plays, or where words Shakespeare invented first appear.