

SUMMARY

Project 3 is an exercise in the implementation and use of hash table and AVL tree data structures. The objective of the project was to do analysis on text documents. The project utilizes hash table and AVL trees to do a selective word count analysis, and analyze the relationship of two differing text documents. One specific use of this is analyzing how an author's work generates a similar signature across his or her works. In the project the signature generated relies on taking the normalized frequencies of selected words in the documents and performing a simple function.

The program has support for two command line functions, WordCount.java which counts the frequencies of "meaningful" (remove words $>1\%$ and $<.01\%$) words in a document. WordCount also supports counting the unique words in a document. Correlator.java does the same thing as WordCount but correlates two documents. Taking the data outputted from WordCount and performing an analysis to see how related two documents are.

QUESTIONS ANSWERED:

- Who is in your group?

Elton Vinh, Thien Van

- How long did the project take?

Project initiated on September 24th, 2015 and ended on October 4th, 2015. Spanned about 11 days. Roughly 8 hours spent developing the data structures, 4 hours developing the word count and correlator algorithms, 1 hour determining and selecting the relevant text files for study, 6 hours developing automated tests, 3 hours writing the report. Total time contributed ~20-25 hours.

- Before you started, which data structure did you expect would be the fastest?

The expectation is that Hash Table would be the fastest. Insertions are $O(1)$ versus $O(n \log n)$ for AVL tree.

- Which data structure is the fastest? Why were you right or wrong?

AVL tree is the fastest for the majority of the test cases run with WordCount.java. We were both right and wrong; it depends on the use case; for super large text files (i.e. The King James Bible) we were right as the Hash Table wins, but for "regular" size text files (like the plays, classic novels, scientific papers), we were wrong as the AVL Tree wins. The regular size files are the typical use case.

- In general, which DataCounter dictionary implementation was "better": trees or hash tables? Note that you will need to define "better" (ease of coding, ease of debugging, memory usage, disk access patterns, runtime for average input, runtime for all input, etc)

In terms of ease of coding the DataCounter ADT implementation of Hash Table was easier to code. The functions were fairly easy to implement and debug because the only case that was different from a normal HashTable was inserting data that already existed in the table and all that required was a simple else if statement to implement. AVL Tree was more difficult because it wasn't simple cut and dry insertion but also the additional implementation of balancing. The implementation of the DataCounter functions however was on par in terms of difficulty.

- Are there cases in which a particular data structure performs really well or badly in the correlator? Enumerate the cases for each data structure.

In general Correlator runtimes are analogous to the WordCount runtimes. The test was run for 12 comparison cases for each of the ADT (BST, AVL, HT). AVL performs the best, followed by BST, followed by HashTable. The runtimes however are very close to each other. A summary is listed below:

AVL: 7.73 seconds

BST: 7.86 seconds

HT: 8.76 seconds

See **Appendix A: Figure 1** for the summary of the test run

- Give a one to two paragraph explanation of whether or not you think Bacon wrote Shakespeare's plays based on the data you collected. No fancy statistical analysis here (formal analysis comes later); keep it fun and simple.

For the test methodology we used multiple works of Shakespeare and Bacon, as well as a couple of other texts for comparison. In order the texts are:

- Shakespeare: hamlet, othello, the-tempest
- Sir Francis Bacon: essays, novum (scientific), the-new-atlantis
- King James Bible
- alice (Alice in Wonderland)

For the Test design:

0) Use HashTable backend for all tests (assuming it's the most efficient)

1) Compare all permutations of Shakespeare's works to find errorSums. Note the median errorSum for these 3 Shakespeare works (expected errorSum when comparing works within single author) which is (3 choose 2). Use the text that produces the median errorSum to compare against others.

2) Do the same for Sir Francis Bacon

3) Run the different permutations (4 choose 2)

s b, s k, s a, b k, b a, k a

Based on the test results, **we believe that they are different authors**. Shakespeare – Shakespeare errorSum is about: $1E-4$. Bacon – Bacon errorSum is about: $4E-5$. Shakespeare – Bacon errorSum is significantly higher, about: $5.6E-4$. For detailed analysis, see **Appendix B**.

- Writeup your benchmarks (this is long). Your mission is to convince us that your benchmark makes sense and that we should be interested in it if we are trying to ascertain which data structure is better suited for your input. You will need to answer at least the following (all formal analysis should answer something similar):

- What are you measuring?

Measuring runtime of the count scripts (shell, perl), and WordCount with the 3 data structures (BST, AVL, HT) by doing direct comparisons on 8 total texts.

- What is the definition of “better” given your measurement?

Better means giving the same results in less time (lower runtime)

- Why is the measurement interesting in determining which is the superior algorithm for the project?

For real life applications, response and calculation time are incredibly important. Searching, insertion, deletions are regular used. Examples include simple sorting algorithms in determining order. If this were a commercial product, customers would value response time highly. Both data structures are able to accomplish the same task so it comes down to which will perform quicker.

- What was your method of benchmarking?

Method of benchmarking was running the programs through a PowerShell script. Each of the 5 count frequency runs (shell, perl, Java BST, Java AVL, Java HT) for each of the 8 texts was timed individually. Frequency is noted for results < 1% or > 0.01 %. Short times are noted. See **Appendix A: Figure 2** for detailed results.

- What were the sources of errors?

Sources of error mostly came from the data structure implementation. As the data structure needed to implement the incCount() method. Also, the initial Hash Table size wasn't efficient for these set of tests. We needed a much larger prime so that resizing (overhead) did not occur as much.

- What were your results?

Results were that for HashTable is the fastest for the large text case (King James Bible). However, for all other cases, AVL Tree won. This is true for both the WordCount and Correlator timing tests. See **Appendix A: Figure 3** for detailed results.

- How did you interpret your results?

Results were mostly interpreted by the runtimes. For the errorSum analysis, basic statistical techniques such as range, mean, median were used to determine “uniqueness” of a work. For Shakespeare and Bacon, the “average” text was used for different test permutations.

- What were your conclusions?

From these texts, for a single author the average errorSum is about $8E-5$ and largest difference of $1.75E-4$. For different authors, there is about a $4E-4$ overall average errorSum and largest range of $8E-4$. There seems to be about a **4 factor average difference between different authors**. The errorSum range for a single both single and different authors are about **2 X factor larger than the average**.

- Are there any interesting directions for future study?

More detailed analysis would need to be used with more precise word stemming techniques. In this case most of the texts used were at least a few hundred years old; more detailed testing would have more texts from the modern era. In addition, text size was not varied; future testing will have short stories, different genres of text, modern scientific papers and larger texts (e.g. the Epic).

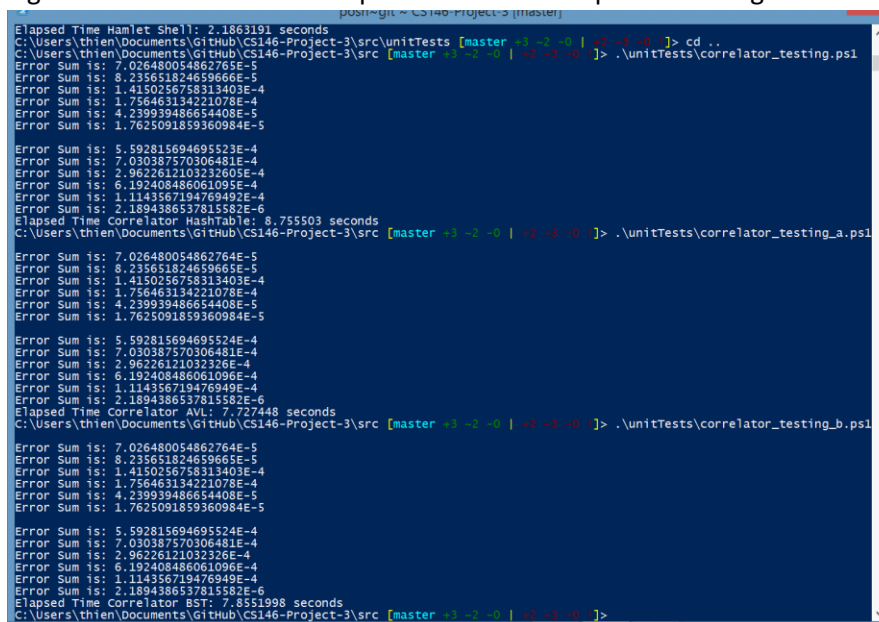
- What did you enjoy about this assignment? What did you hate? Could we have done anything better?

We enjoyed the having the flexibility to perform our own analysis. Picking the texts for our cases was fun. Coming up with the automated testing is a good skill to have.

One thing we did not like is the assumed reliance on Unix/Linux systems for running the timings. To improve, the instructors can have additional instructions for Windows users (e.g. basics of powershell timing).

Appendix A: Raw results

Figure 1: Correlator time comparisons for 12 comparisons using HashTable, AVL, BST respectively



```
Elapsed Time Hamlet Shell: 2.1863191 seconds
C:\Users\thien\Documents\GitHub\CS146-Project-3\src\unitTests [master v3 -2 -0 | v2 -3 -0 ]> cd ..
C:\Users\thien\Documents\GitHub\CS146-Project-3\src [master v3 -2 -0 | v2 -3 -0 ]> .\unitTests\correlator_testing.ps1
Error Sum 1s: 7.026480054862764E-5
Error Sum 1s: 8.235651824659665E-5
Error Sum 1s: 1.4150256758313403E-4
Error Sum 1s: 1.756463134221078E-4
Error Sum 1s: 4.239939486654408E-5
Error Sum 1s: 1.7625091859360984E-5

Error Sum 1s: 5.592815694695523E-4
Error Sum 1s: 7.030387570306481E-4
Error Sum 1s: 2.9622612103232605E-4
Error Sum 1s: 6.192408486061096E-4
Error Sum 1s: 1.1143567194769492E-4
Error Sum 1s: 2.1894386537815582E-6
Elapsed Time Correlator HashTable: 8.755503 seconds
C:\Users\thien\Documents\GitHub\CS146-Project-3\src [master v3 -2 -0 | v2 -3 -0 ]> .\unitTests\correlator_testing_a.ps1
Error Sum 1s: 7.026480054862764E-5
Error Sum 1s: 8.235651824659665E-5
Error Sum 1s: 1.4150256758313403E-4
Error Sum 1s: 1.756463134221078E-4
Error Sum 1s: 4.239939486654408E-5
Error Sum 1s: 1.7625091859360984E-5

Error Sum 1s: 5.592815694695524E-4
Error Sum 1s: 7.030387570306481E-4
Error Sum 1s: 2.96226121032326E-4
Error Sum 1s: 6.192408486061096E-4
Error Sum 1s: 1.114356719476949E-4
Error Sum 1s: 2.1894386537815582E-6
Elapsed Time Correlator AVL: 7.727448 seconds
C:\Users\thien\Documents\GitHub\CS146-Project-3\src [master v3 -2 -0 | v2 -3 -0 ]> .\unitTests\correlator_testing_b.ps1
Error Sum 1s: 7.026480054862764E-5
Error Sum 1s: 8.235651824659665E-5
Error Sum 1s: 1.4150256758313403E-4
Error Sum 1s: 1.756463134221078E-4
Error Sum 1s: 4.239939486654408E-5
Error Sum 1s: 1.7625091859360984E-5

Error Sum 1s: 5.592815694695524E-4
Error Sum 1s: 7.030387570306481E-4
Error Sum 1s: 2.96226121032326E-4
Error Sum 1s: 6.192408486061096E-4
Error Sum 1s: 1.114356719476949E-4
Error Sum 1s: 2.1894386537815582E-6
Elapsed Time Correlator BST: 7.8551998 seconds
C:\Users\thien\Documents\GitHub\CS146-Project-3\src [master v3 -2 -0 | v2 -3 -0 ]>
```

Figure 2: Error sums of the various permutations

```

posh~git ~ CS146-Project-3 [master]
Match with word: bring freq1 = 1.1634917444972129E-4, freq2 = 1.603249251817016E-4
Match with word: circumstances freq1 = 1.1634917444972129E-4, freq2 = 1.0688328345446772E-4
Match with word: org freq1 = 1.1634917444972129E-4, freq2 = 1.3360410431808466E-4
Match with word: avoid freq1 = 1.057719767724739E-4, freq2 = 1.0688328345446772E-4
Match with word: refund freq1 = 1.057719767724739E-4, freq2 = 1.2469716403021235E-4
Match with word: seeing freq1 = 1.057719767724739E-4, freq2 = 1.1579022374234003E-4
Match with word: behind freq1 = 1.057719767724739E-4, freq2 = 1.0688328345446772E-4
Match with word: forces freq1 = 1.057719767724739E-4, freq2 = 1.0688328345446772E-4
Match with word: forth freq1 = 1.057719767724739E-4, freq2 = 2.493943280604247E-4
Match with word: sources freq1 = 1.057719767724739E-4, freq2 = 1.0688328345446772E-4
Match with word: took freq1 = 1.057719767724739E-4, freq2 = 1.5141798489382927E-4
Match with word: extreme freq1 = 1.057719767724739E-4, freq2 = 1.0688328345446772E-4
Match with word: danger freq1 = 1.057719767724739E-4, freq2 = 1.1579022374234003E-4
Match with word: person freq1 = 1.057719767724739E-4, freq2 = 1.9595268633319083E-4
Match with word: learned freq1 = 1.057719767724739E-4, freq2 = 1.1579022374234003E-4
Match with word: altogether freq1 = 1.057719767724739E-4, freq2 = 1.2469716403021235E-4
Match with word: prepare freq1 = 1.057719767724739E-4, freq2 = 1.0688328345446772E-4
Match with word: possible freq1 = 1.057719767724739E-4, freq2 = 1.1579022374234003E-4
Match with word: heard freq1 = 1.057719767724739E-4, freq2 = 1.1579022374234003E-4
Match with word: side freq1 = 1.057719767724739E-4, freq2 = 1.3360410431808466E-4
Match with word: letters freq1 = 1.057719767724739E-4, freq2 = 1.3360410431808466E-4
Match with word: appeared freq1 = 1.057719767724739E-4, freq2 = 1.1579022374234003E-4
Match with word: additional freq1 = 1.057719767724739E-4, freq2 = 1.0688328345446772E-4
Match with word: head freq1 = 1.057719767724739E-4, freq2 = 1.3360410431808466E-4
Error Sum is: 1.7625091859360984E-5
Total words in document 1: 94543
Total words in document 2: 112272
C:\Users\thien\Documents\GitHub\CS146-Project-3\src [master v5 -> v1 ]> javac -cp .\wordcounter\Correlator.java
C:\Users\thien\Documents\GitHub\CS146-Project-3\src [master v5 -> v1 ]> .\unitTests\correlator_testing.ps1
Error Sum is: 7.026480054862765E-5
Error Sum is: 8.235651824659666E-5
Error Sum is: 1.4150256758313403E-4
Error Sum is: 1.756463134221078E-4
Error Sum is: 4.23939486654408E-5
Error Sum is: 1.7625091859360984E-5
C:\Users\thien\Documents\GitHub\CS146-Project-3\src [master v5 -> v1 ]> .\unitTests\correlator_testing.ps1
Error Sum is: 7.026480054862765E-5
Error Sum is: 8.235651824659666E-5
Error Sum is: 1.4150256758313403E-4
Error Sum is: 1.756463134221078E-4
Error Sum is: 4.23939486654408E-5
Error Sum is: 1.7625091859360984E-5
Error Sum is: 5.592815694695523E-4
Error Sum is: 7.030387570306481E-4
Error Sum is: 2.9622612103232605E-4
Error Sum is: 6.192408486061095E-4
Error Sum is: 1.1143567194769492E-4
Error Sum is: 2.1894386537815582E-6
C:\Users\thien\Documents\GitHub\CS146-Project-3\src [master v5 -> v1 ]>

```

Figure 3: Runtimes of WordCount.java for AVL and HashTable

Elapsed Time Hamlet Java-AVL: 0.294674 seconds	
Elapsed Time Othello Java-AVL: 0.2735111 seconds	
Elapsed Time The-Tempest Java-AVL: 0.3277355 seconds	
Elapsed Time Essays Java-AVL: 0.3297796 seconds	
Elapsed Time Novum Java-AVL: 0.3699642 seconds	
Elapsed Time The-New-Atlantis Java-AVL: 0.3146989 seconds	
Elapsed Time King-James-Bible Java-AVL: 0.6582965 seconds	
Elapsed Time Alice Java-AVL: 0.2806521 seconds	
Elapsed Time Hamlet Java-HashTable: 0.3126671 seconds	
Elapsed Time Othello Java-HashTable: 0.3066474 seconds	
Elapsed Time The-Tempest Java-HashTable: 0.4184704 seconds	
Elapsed Time Essays Java-HashTable: 0.3500107 seconds	
Elapsed Time Novum Java-HashTable: 0.3993812 seconds	
Elapsed Time The-New-Atlantis Java-HashTable: 0.3429035 seconds	
Elapsed Time King-James-Bible Java-HashTable: 0.6277829 seconds	
Elapsed Time Alice Java-HashTable: 0.2979477 seconds	

Appendix B: Raw analysis

error Sum for Shakespeare, Bacon							
S							
lowest two are first 2 tests							
range is $7E-5$							
average is about $1E-4$							
median is hamlet							
B							
lowest are the first and 3rd							
range is $1.2E-4$							
average is about $4E-5$							
median is novum							
within a single author, the largest difference is $1.75E-4$ with the average about half of that							

surprisingly, alic and bible are remarkably similar in writing style (based on our methodology)							
upon further research, Lewis Carroll (author of Alice in Wonderland) was a Christian cleric							
may contribute to the similarities							
$3.8E-4$ overall average							
$4.6E-4$ average difference without alic/bible outlier							
$6.3E-4$ without alic at all							
$3E-4$ shakespeare and alic							
largest difference about $7E-4$ (4 X than within an author)							
overall average about half of that							