CSE 373 Homework 4 Write up

- **Who is in your group (Give name, UW NetID & student number of each person)?**

Geoff Gray gegray 1463717
Austin Meyers arm38 1228316

- **a) How did you design your tests & what properties did you test?**

We designed the tests to check the methods that we implemented.
Both HashTable implementations were tested to check if the hashing functions were working correctly, proper edge case handling, and expected output.
We also tested Correlator, which gave the correct output, and handled input correctly.
StringComparator was also tested with edge cases and other input and returned expected values.
StringHasher was tested in using the above classes and methods.

b) What boundary cases did you consider?

We considered the following edge cases:
A file with no text -> WordCount, Correlator, StringComparetor gave expected outputs
A file with more text than the primes list -> HashTable: See an index out of bound exception, add to primes list
Two equal string -> StringComparator gives expected output

- **Conduct an experiment to determine which `DataCounter` implementation (HashTable_SC, HashTable_OP) is better for large input texts.**

a) Describe your experimental setup:

1) Inputs used

2k.txt (2,000 words), 20k.txt (20,000 unique words), Hamlet (200,000 words), dictionary.txt (400,000 unique words)

2) How you collected timing information

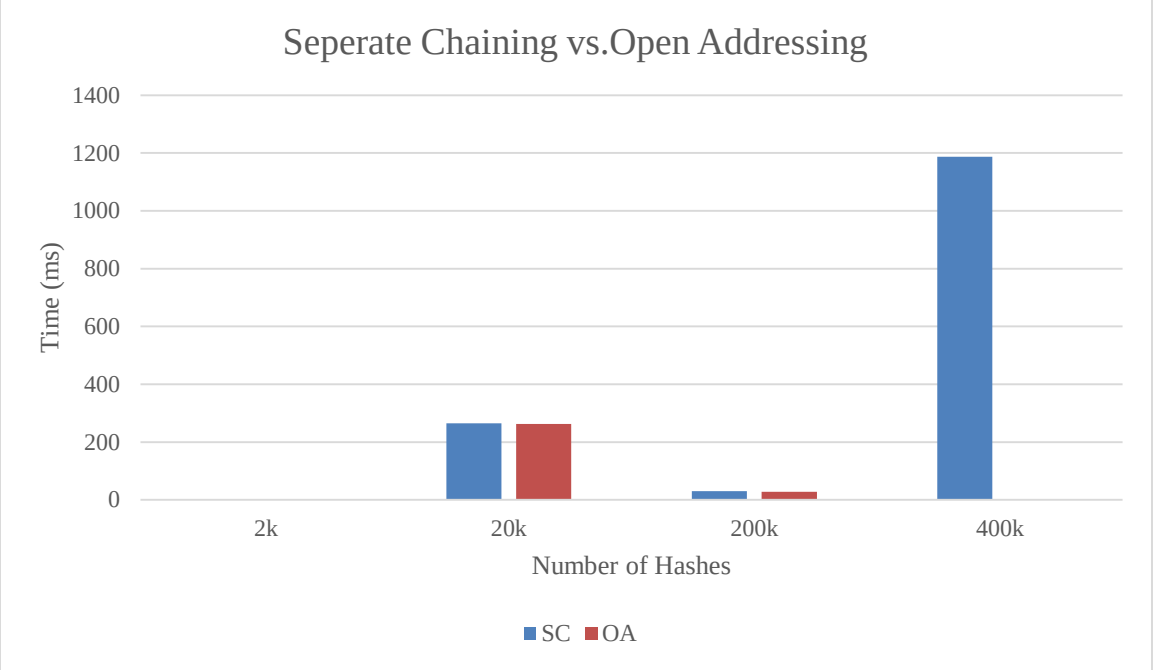
Primary testing was performed on Hamlet (~200,000 hashes) & dictionary.txt (~400,000 hashes) using both SC and OA. The tests were run 100 times, ignoring the first 10 passes (JVM "warm-up") and then we took the average of the remaining 90 tests. Results are reported in milliseconds.

3) Any details that would be needed to replicate your experiments

No, all information is included in our submitted code. I suppose we should mention that tests were run on different machines, both of which were Apple laptops, though with different specs.

b) Experimental Results (Place your graphs and tables of results here).

	2k hashes	20k hashes	200k hashes	400k hashes
SC	2.84444444446 ms	265.24444447 ms	30.077777 ms	1187.75555 ms
OA	2.72222222223 ms	263.08888889 ms	28.188889 ms	Infinite



c) Interpretation of Experimental Results

1) What did you expect about the results and why?

We would expect Separate Chaining to be a bit slower than open addressing. This is because all find() insert() and delete() operations have to go through a linked list which has a complexity of O(n). We would venture to guess that Open Addressing may get complex if there are a lot of collisions.

2) Did your results agree with your expectations? Why or Why not?

No, Open Addressing seems to be more efficient in our testing cases. Attempting to analyze the cause, we concluded that the insert() and find() functions for a Separate Chaining Hash Table would be very slightly slower in most cases, as you have to traverse a linked list. With a large list of completely unique hashes, the Separate Chaining should be faster (only single nodes), but this is not a likely test case.

3) According to your experiment, which Hashtable implementation, separate chaining or open addressing, is better?

In almost all cases, the Open Addressing implementation is a more efficient algorithm. If the input is VERY large, and each hashed value is unique, then Open Addressing starts to become far less efficient. I don't imagine using a Hash Table to create a dictionary of unique words is a common use case, so overall we would recommend using Open Addressing when using a Hash Table.

- **Conduct experiments to determine if changing the hash function affects the runtime of your HashTable.**

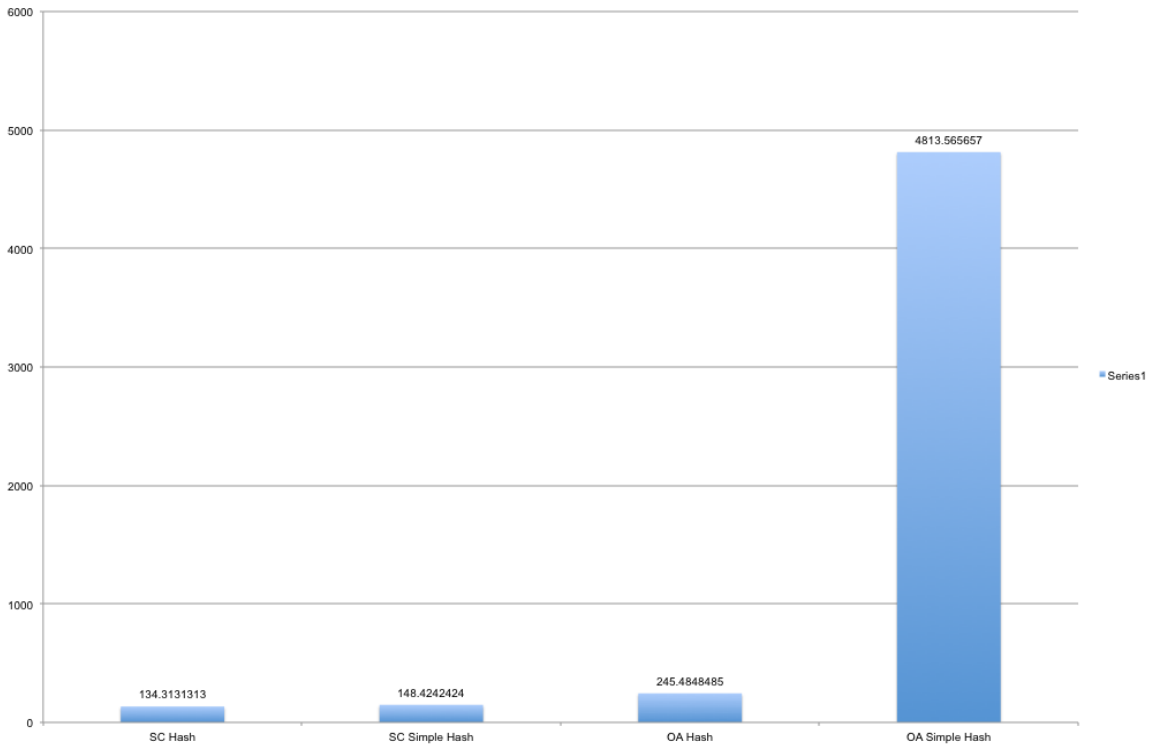
a) Brief description of your hash functions

Good Hash: The sum of ((distance of each character in the string from 'a') * 3) % 269683
Bad Hash: The sum of (distance of each character in the string from 'a')

b) Experimental Results (Place your graphs and tables of results here).

Experiment with at least 2 hash functions (2 Hashing functions = 2 experiments depending on how you measured the runtime)

Don't forget to give each graph a title and label the axes.



All values in ms.

c) Interpretation (Your expectations and why? Did it match your results? If not, why?)

We expected the good hash function to outperform the simple (bad) hash function. We were surprised to see that SC handled the simple hash so well, especially compared to OA. In retrospect, it does seem to make sense because the simple hash likely produces many duplicate hash values which then must all be traversed with each subsequent insert. Seeing these results makes us wonder if there is an alteration we could make to the good hash function to make it even better, perhaps by multiplying by the string's first char value or its length, something more unique than 3.

- **Using `Correlator`, does your experimentation suggest that Bacon wrote Shakespeare's plays?**

Show at least one (you can experiment with more texts if you want) correlation value for each of:
a) Shakespeare's work compared to Shakespeare's work

hamlet / hamlet
0.0
othello / the-tempest
2.688637953675988E-4
othello / hamlet
1.7655777913126052E-4
the-tempest / hamlet
2.6693013476318083E-4

b) Bacon's work compared to Bacon's work

the-new-atlantis / the-new-atlantis
0.0
the-essays / the-advancement-of-learning
1.344086629975853E-4
the-essays / the-new-atlantis
3.922436403822705E-4
the-advancement-of-learning / the-new-atlantis
3.805931402277071E-4

c) Shakespeare's work compared to Bacon's work

hamlet / the-new-atlantis
5.65727366923397E-4
hamlet / the-advancement-of-learning
4.579971811775332E-4
hamlet / the-essays
5.621769030248902E-4
othello / the-new-atlantis
6.682467247590484E-4
othello / the-advancement-of-learning
5.395547385899665E-4
othello / the-essays
6.463729850450667E-4
the-tempest / the-new-atlantis
5.039603559242515E-4
the-tempest / the-advancement-of-learning
3.122391221109452E-4
the-tempest / the essays
4.3335173590321717E-4

According to the results of your experiments, did Bacon write Shakespeare's plays?

No, the correlation value for Bacon vs. Shakespeare signatures is fairly low, when you compare that value with the correlation value of the authors writings vs. the authors other writings.
It is interesting to note that the correlation value for Shakespeare's 'The Tempest' vs. Bacon's writings has the closest correlation value.

- **Include a description of how your project goes "above and beyond" the basic requirements (if it does).**

Testing numerous texts from each other gives us a wider set of signatures to compare, making our analysis more precise. Also, testing edge cases (~400,000 word dictionary vs document containing ~400,000k of the same word. Also read wikis to better understand the advantages of SC and OA and vice versa.

- **If you worked with a partner:**

a) Describe the process you used for developing and testing your code. If you divided it, describe that. If you did everything together, describe the actual process used (eg. how long you talked about what, what order you wrote and tested, and how long it took).

We used version control systems to develop our code, and wrote unit tests on our own. The timing and comparison tests for the write up were written as a team. We split up the tasks between the 2 of us, each of us writing one of the Hash Table implementations.

b) Describe each group member's contributions/responsibilities in the project.

Geoff : StringComparator, HashTable_OA, Correlator, WriteUp, testing
Austin : WordCount, StringHasher, HashTable_SC, WriteUp, testing

c) Describe at least one good thing and one bad thing about the process of working together.

Bad: Different development environments, IDE's
Good: Collaborate on ideas and share understanding, error spotting, analysis (having two sets of eyes led to much better analysis)

- **a) Which parts of the project were most difficult?**

Gaining and understanding of all the moving parts of this assignment was difficult. There felt to be numerous portions of the project that required an understanding of another piece.

b) How could the project be better?

Discussing the project more in class and section. This was the biggest and most complicated assignment thus far, and it would have helped to discuss the subject matter more in depth. Improving comments for shake_n_bacon would be helpful as the provided code contained some of the code needed in those classes. Let us use just a few built in methods (writing a string comparator was irrelevant to this project).

Appendix

Place anything else that you want to add here.