CIS 1210 66892455

# Homework 09: Quantitative Analysis

Collaborators: None

---

**Q1. Output of TestHarness.java**

```
            dictionary.txt
--------------
10 iterations complete.
20 iterations complete.
30 iterations complete.
HashMap: Average running time: 61607 microseconds
HashMap: Average memory usage: 30484 kilobytes
10 iterations complete.
20 iterations complete.
30 iterations complete.
TrieMap: Average running time: 176509 microseconds
TrieMap: Average memory usage: 99045 kilobytes
10 iterations complete.
20 iterations complete.
30 iterations complete.
java.util.HashMap: Average running time: 62064 microseconds
java.util.HashMap: Average memory usage: 33218 kilobytes
10 iterations complete.
20 iterations complete.
30 iterations complete.
java.util.TreeMap: Average running time: 1006882 microseconds
java.util.TreeMap: Average memory usage: 33903 kilobytes

phonenumbers.txt
-------------
10 iterations complete.
20 iterations complete.
30 iterations complete.
HashMap: Average running time: 952 microseconds
HashMap: Average memory usage: 761 kilobytes
10 iterations complete.
20 iterations complete.
30 iterations complete.
TrieMap: Average running time: 1187 microseconds
TrieMap: Average memory usage: 331 kilobytes
10 iterations complete.
20 iterations complete.
```

```
30 iterations complete.
java.util.HashMap: Average running time: 1278 microseconds
java.util.HashMap: Average memory usage: 824 kilobytes
10 iterations complete.
20 iterations complete.
30 iterations complete.
java.util.TreeMap: Average running time: 34017 microseconds
java.util.TreeMap: Average memory usage: 824 kilobytes

Process finished with exit code 0
```

**Q2. Run Time and Space Usage for dictionary.txt and phonenumbers.txt**

For dictionary.txt, which implementation (between your implementations of TrieMap and HashMap) had a better running time? Which implementation had better space usage? What about for phonenumbers.txt? Was this what you were expecting? Why or why not?

For dictionary.txt, the HashMap implementation had a better run time; the run time of HashMap was 61607 microseconds, while the run time of TrieMap was 176509 microseconds. HashMap implementation had a better space usage; the space usage of HashMap was 30484 kilobytes, while the space usage of TrieMap was 99045 kilobytes.

For phonenumbers.txt, HashMap implementation had a better run time; the run time of HashMap was 952 microseconds, while the run time of TrieMap was 1187 microseconds. TrieMap implementation had a better space usage; the space usage of TrieMap was 331 kilobytes, while the space usage of HashMap was 761 kilobytes.

I expected HashMap to have a better run time for both, as HashMaps usually have better runt times than TrieMaps. I also expected the space usage of TrieMap to be better for both, especially phonenumbers.txt, due to the area code prefixes.

**Q3. Space Efficiency of Tries and Hash Tables**

This was reflected in phonenumbers.txt but not dictionary.txt. This was likely because tries use prefixes for their efficiency. Because phone numbers have area codes, there are more specific prefixes than the many random words in dictionary.txt that are probably less likely to share a common prefix. You could potentially improve the memory consumption of the TrieMap by adding common consonant blends (sh, ch, schw, etc) and diphthongs (ie, ea, etc.) as keys, which would make prefix of length 2 or 3 into a prefix of length 1.

**Q4. Big-O Notation and Run Time/Space Usage**

Big-Oh notation, which is the upper bound, does not necessarily tell us much that is relevant for actual running time or space usage for an algorithm on a data set. This is because although it is an upper limit, the actual ("expected") run time can vary greatly, depending on the type and makeup of the data set. The implications for software development are that it is important to consider the specific data set that you will be working with when developing an algorithm, as certain types of data sets will lend themselves to behaving more efficiently.

**Q5.** The run time and space usage for TrieMap and dictionary.txt with initChildren() are 119950 microseconds and 132280 kilobytes, respectively, compared to the original 176509 microseconds and 99045 kilobytes; thus, the lazy iteration

increases the run time by 56559 microseconds and decreases the space usage by 33235 kilobytes., the lazy iteration is a worthwhile optimization, as it saves a substantial amount space without increasing the time by that much.

**Q6. Comparisons with Java's HashMap Implementation**

For both dictionary.txt and phonenumbers.txt, my HashMap had a very similar but slightly faster run time and also similar but slightly better space usage. For phonenumbers.txt, TrieMap had a better run time and better space usage. My implementation was likely a bit more simple and specifically tailored to this than the util.Java implementation, which also may contain extraneous things that I did not need in my HashMap.

**Q7. Comparisons with Java's TreeMap Implementation**

For dictionary.txt, the TrieMap had a better run time but worse space usage tha TreeMap. For phonenumbers.txt, TrieMap had a better run time and better space usage. TreeMap and java.util.HashMap had very similar space usage for both dictionary.txt and phonenumbers.txt, but TreeMap's run time was substantially faster. This is because HashMaps are designed to reduce collisions and therefore improve run time, while TreeMaps require a significant amount of time to traverse the tree.