Project 7: Comparing Data Structures

Abstract

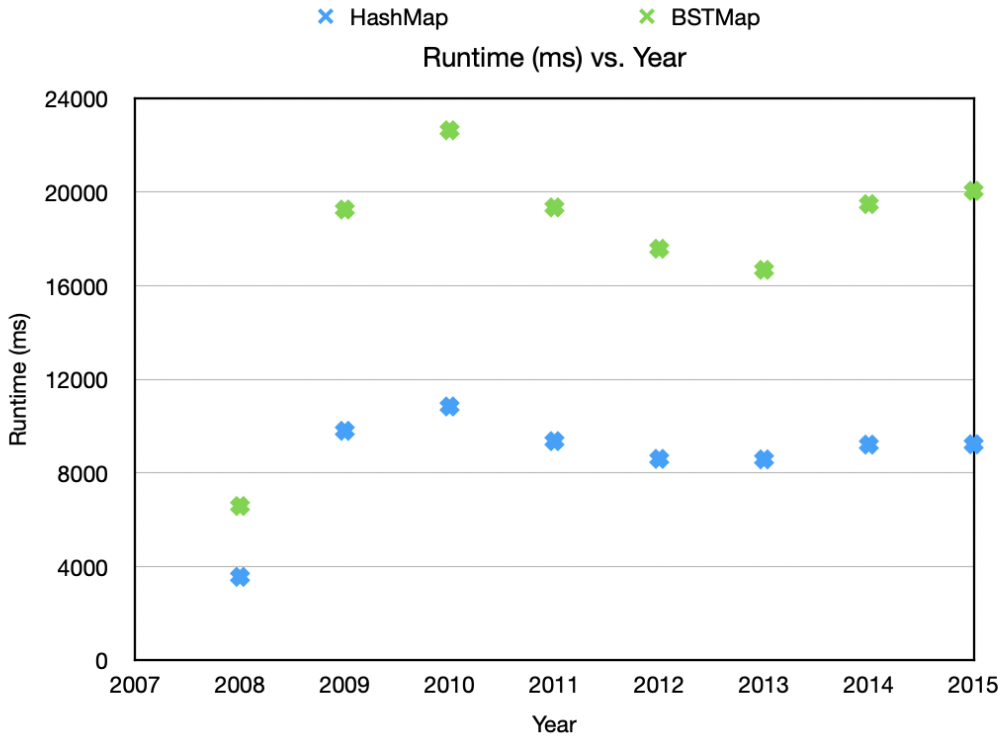
In this week's project, I implemented my own chaining HashMap. It is a data structure used to store key and value pairs and like last week's BSTMap, it implements from MapSet interface. In fact, this week continues to find the word frequencies from the reddit comments files, but using two data structures now. I created a WordCounter2 class to compare the efficiency of HashMap and that of BSTMap. First, I separated the I/O process by using `readWords()` method; then, I used `buildMap()` to build two maps from the word list and measured the time taken for each data structure. The result of my project shows that HashMap has a better efficiency in the building map field.

Result

The result of the run time is calculated from five repetition of building maps from the ArrayList of words. I compared the run-time taken for these two different data structures to build the map.

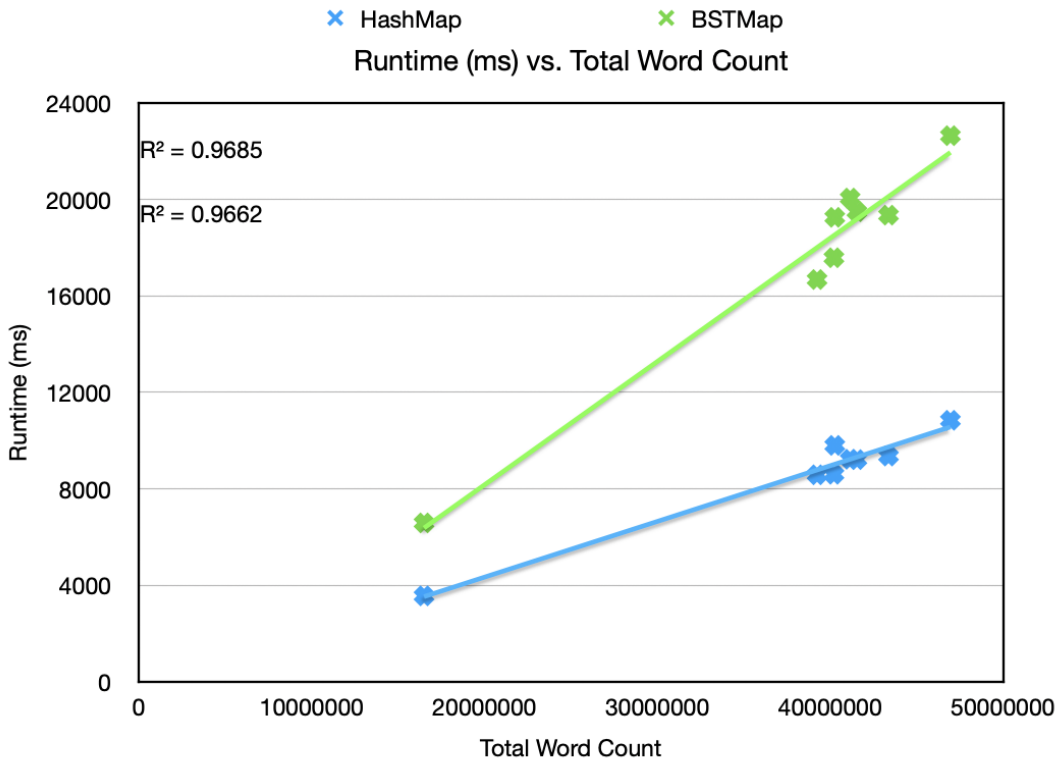
Three Graphs

The following graph shows the relationship between year and runtime in millisecond.



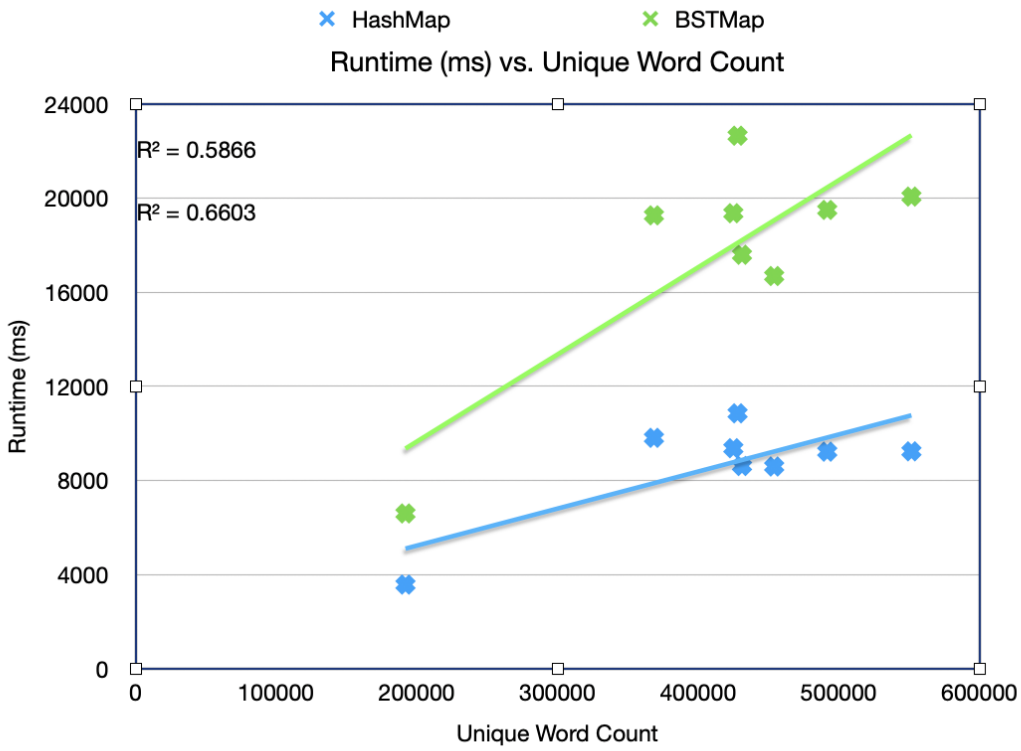
From above, we can see HashMap takes less time in all the years of text files compared to BSTMap. They both share the same trend tho - they run faster in 2008, and slower in other years.

The next graph shows the relationship between total word counts and runtime in ms.



We can see that, building maps from the same amount of words, HashMap takes much less time than BSTMap. They both take longer time when the total word count increases.

The next graph shows the relationship between unique word counts and runtime in ms.



In this graph, although we can see a higher efficiency of HashMap compared to BSTMap, the relationship of unique word count of run time is not as obvious as the last graph, shown from the low R square value.

These plots make senses to me. Since HashMap has Order 1 of accessing time and inserting time, it performs faster than BSTMap.

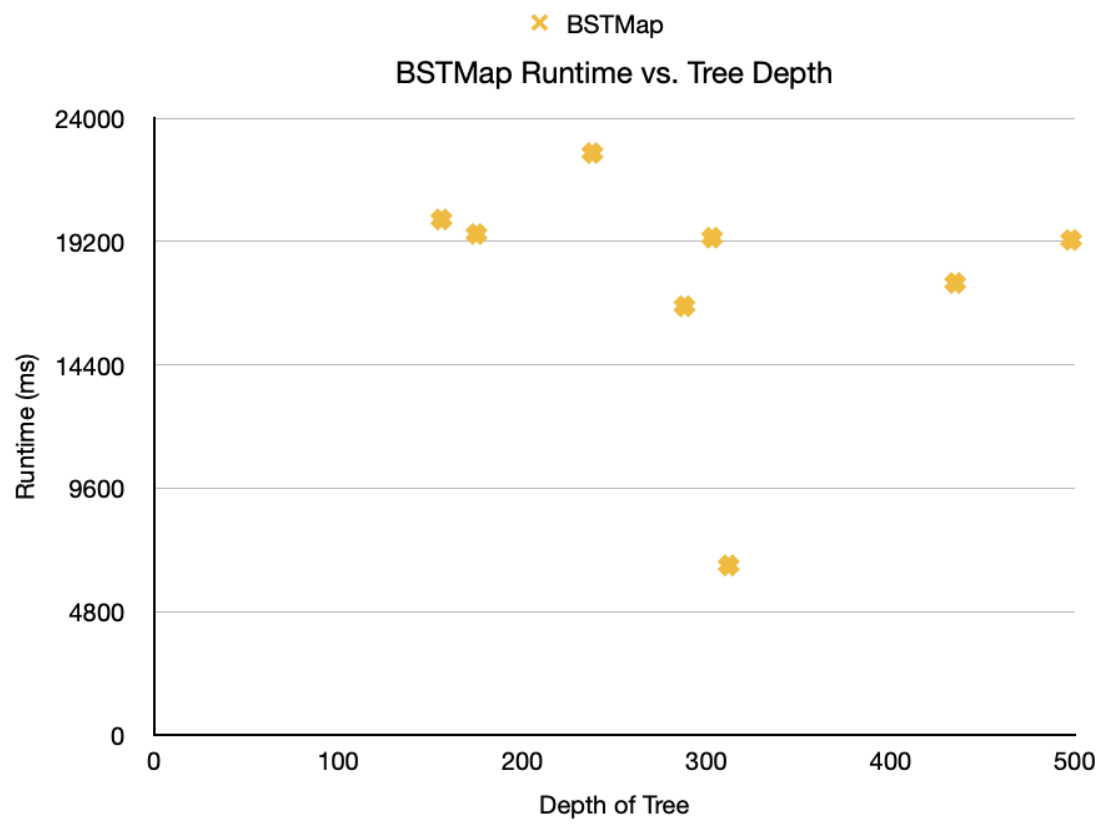
Analysis of the Performance

When analyzing the performance between these two map structures, one essential thing to take into consideration is the `get()` and `put()` method. In face, inside the `buildMap()` class, it is these two functions that determine the performance of building the maps. The higher efficiency these two functions has, the higher efficiency it performs in building the maps. For HashMap, it has O(n) as the worst case behavior and O(1) as the average case for both methods, while for BSTMap, it has O(n) as the worst case, and O(log n) as the average for both methods. Since O(1) is faster than O(log n), we see HashMap performs better than BSTMap.

In terms of being close to ideal, we can see from the graph above that shows the relationship between total word counts and run time, the BSTMap shows a linear trend, which is not the ideal situation where the time complexity should

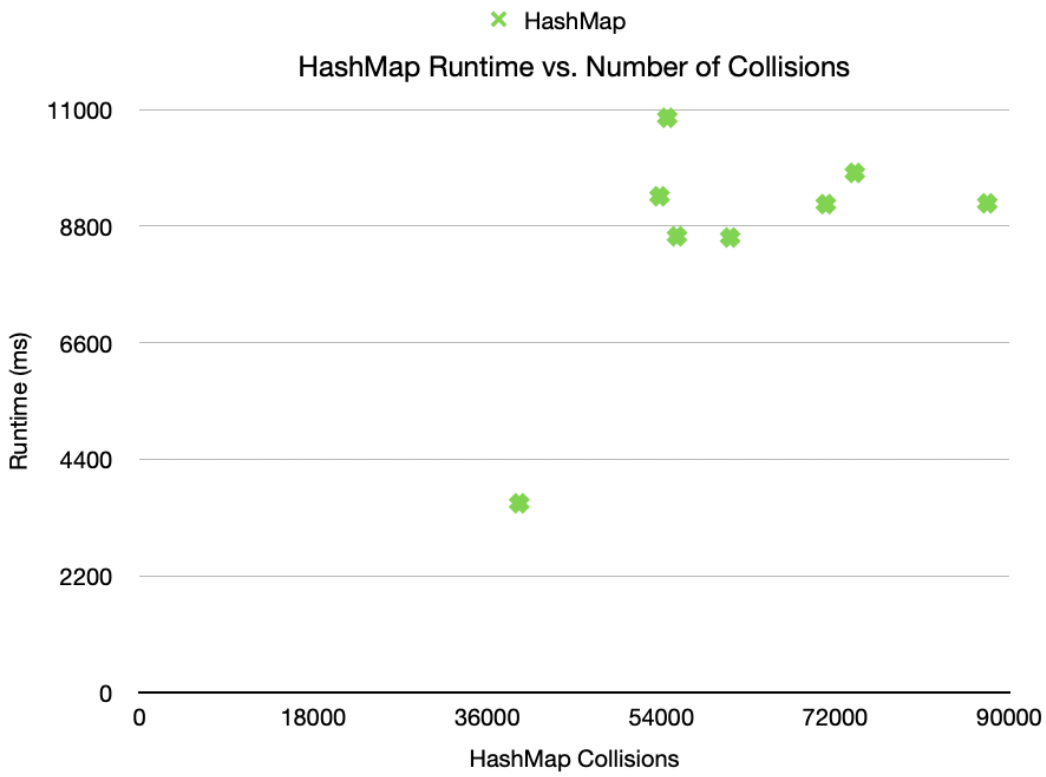
be O(log n). The HashMap shows a ideal case when total word counts is around 40 million; however, it generally shows a growing trend between the total word count and the run time.

The following graph shows the relationship between the depth of tree and the runtime of BSTMap after the map is built.



We can see that the depth of tree does not necessarily affect runtime.

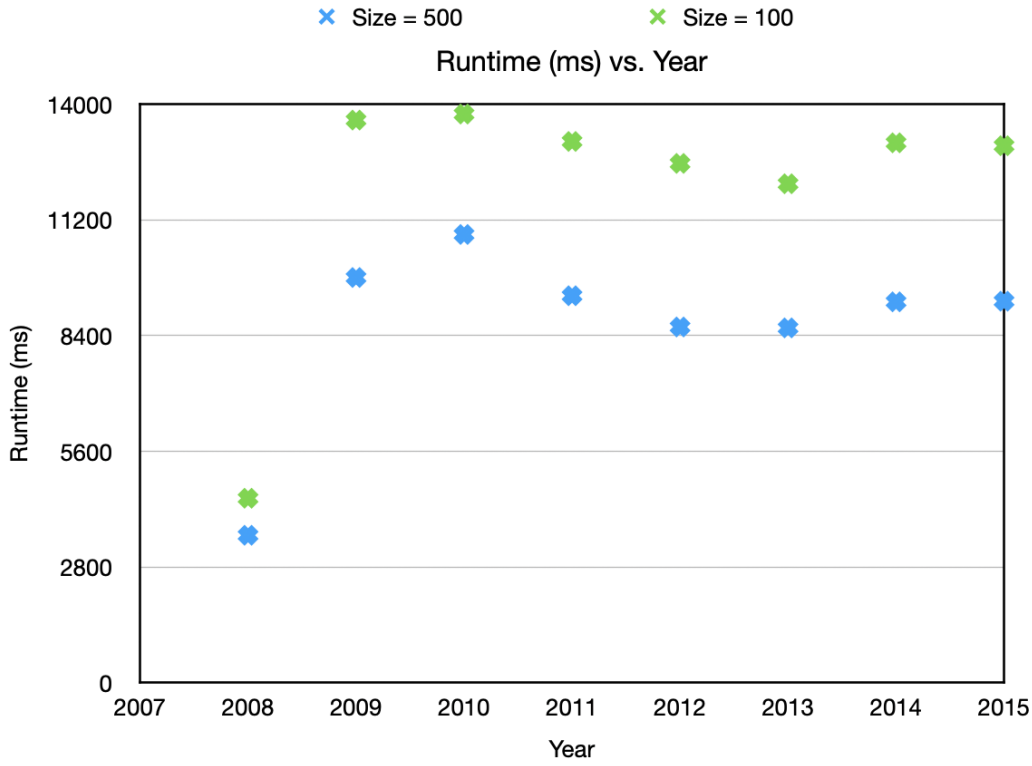
The following graph shows the relationship between the number of collisions and the runtime of HashMap.



We can see that the number of collisions does not necessarily affect the runtime of HashMap.

Experimenting Modification on HashMap

In this experiment, I changed the initial size of the Object array in the HashMap from 500 to 100, and the following graph shows the change in runtime.



We see that, after the change in initial size, the HashMap needs more time to build the map from the word list. This makes sense to us because a smaller size of array means more collisions when inserting new elements, thus, it would take more time and lower the efficiency.

Reflections

I learnt the following.

1. I got to know more about handling the collisions in HashMap.
2. I got to experiment modifying the HashMap and see how it affects the performance.