Sam Lee

6/25/18

CSC 130

Professor Chidella

Write-Up Questions

1. Who is in your group?

Myself

2. How long did the project take?

Two weeks

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

AVL Tree

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

The Hash Table implementation yielded the fastest result. The reason we believed that the AVL Tree would be the fastest implementation is because the balancing natural of it. However, the Hash Table implementation seemed to be the fastest. This is because the AVL Tree’s implementation utilizes recursion to insert which slowed it down. In fact, given the BST’s implementation, the AVL Tree’s implementation is slower as well.

In general, which DataCounter dictionary implementation was "better": trees or hashtables? 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).

Hash table was better implementation in relation to ease of coding, debugging and

average runtime of all inputs.

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

NOTE: The times mentioned below are results of taking the time before and after running correlator and printing the words to the screen. This is why the elapsed time is greater than that of executing the countWords function (hereafter referred to as the WordCount function).

When using correlator on Shakespeare’s Hamlet and Sir Francis Bacon’s The New Atlantis, the results of the three implementations were identical. Each implementation resulted in a difference metric of 0.003457. However, the Hash Table performed faster than the BST and AVL Tree,

HT = 326.8ms

BST = 340ms

AVL = 351.4ms

This follows the overall performance of the WordCount function’s implementation with the Hash Table being the fastest.

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.

Our data from correlator revealed that the difference metric was 0.0034 regardless of the data structure used. This result was found through the exemption of common words such the, and, etc. The words that appeared in both documents and their frequencies were normalized respectively and the previously stated difference metric was calculated. This difference metric suggests that Shakespeare’s play was not written by Bacon as the correlation between them was small.

Benchmarking:

What are you measuring?

|  |  |  |  |
| --- | --- | --- | --- |
| *Trials* | *BST (ms)* | *AVL (ms)* | *HASH (ms)* |
| 1 | 66 | 79 | 59 |
| 2 | 66 | 62 | 62 |
| 3 | 68 | 62 | 62 |
| 4 | 68 | 78 | 63 |
| 5 | 71 | 78 | 63 |

|  |  |  |  |
| --- | --- | --- | --- |
| *Average* | 67.8 | 71.8 | 61.8 |

NOTE: The measurements listed above are a result of running the WordCount function.

function without printing the final counts to the screen. This allowed our timer

to ignore the time it would take to print the information to the screen.

We were measuring the runtime of WordCount using the 3 data structures (BST,

AVL and Hash Table). This was done by taking the system time before the call to

the WordCount function and taking the time after its execution. The total time

taken was computed by subtracting the second time from the first

measurement. This was repeated for a total of five time with each data structure

and an average was calculated as shown above.

6. What is the definition of "better" given your measurement?

Better means which data structure implementation gave the result the fastest (took the least amount of time to execute). As can be seen in the table above, the five trials were taken into consideration to find out the overall performance of the WordCount function with the respective data structure. The Hash Table’s performance average is the lowest. This means it performed better than the others.

7. Why is the measurement interesting in determining which is the superior algorithm for this project?

It is interesting to know how fast algorithm can find similar thousand words in few milli-seconds. Working on different algorithms/implementations gives insight of how system works and how well it may be optimized if the results are a sign of underperformance. This was the case for our AVL tree where the performance was staggered by the decision to implement a recursive function to add data into the tree. Trough measurement, we were able to observe this inefficiency and has we had more time, could have optimized its implementation further. Consider the real world, a program’s speed is important to many consumers as they may have a lot more data to process. This is why calculating these measurements is important as it allows for an understanding of how optimized one’s implementation is.

8. What was your method of benchmarking?

As mentioned in an above question, we used java’s System.currentTimeMillis() to record the time before calling the WordCount function and got the difference in time after the functions execution. Psudeocode below (assume timer is a double);

Timer = get current time

Call WordCount function

Timer = get current time – timer

After the execution, timer, in the last assignment statement, would be updated to reflect the elapsed time before and after the call to the WordCount function. As mentioned above, the time was tested without printing any data out.

9. What were the sources of errors?

Most of the errors occurred during the implementation of the data structures (AVL and Hash Table). However, these were the result of inaccurate coding and debugging helped correct them. Initially, only the first character was considered when calculating the index of a word for the Hash Table, however, this was later corrected to utilize the entire word and calculate the index from there.

10. What were your results?

Most of the results focus on the timing as well as the correlation of the two documents. The Hash Table had an average time of 61.8ms which was faster than the other two. The BST followed with 67.8ms and AVL, through the inefficiency of recursion, having the longest time of 71.8ms. The correlator calculated the difference metric between the documents.

11. How did you interpret your results?

Our result was quite different than our anticipation. But that was a learning process and learned some different ways to approach a problem. Our results were interpreted by the time taken for the WordCount function. As stated in the previous questions, the results for our implementation reflect how optimized they are. Despite this, both my partner and I agree that the AVL Tree’s runtime was staggered by using recursion in the intCount function.

12. What were your conclusions?

Through our benchmarks and word correlator, we can conclude that the Hash Table, with chaining, outperformed the other data structure implementation despite not having an O(1) insert time. This was because a certain word may have had to traverse the index it fell into k number of times before reaching the end and being inserted. Regardless, the Hash Table’s implementation reflected as the fastest amongst the three. As for the correlator results, we can conclude that Sir F. Bacon did not write Shakespeare’s Hamlet and that the two documents are not extremely similar given the method used to normal the word frequencies in each document before changing the difference metric as well as excluding a normalized frequency greater than 0.01 and less than 0.0001

12. Are there any interesting directions for future study?

With respect to the optimization of each implementation. Sam and I discussed trying to optimize the code of each data structure to see how the current results vary between the two version of this program. Sadly, we did not have time to do that before this project’s due date. Another thing that could be done to improve the accuracy of the program relates to the correlator and the acceptable range for the normalized frequency of words. Finding the optimum range would most likely yield a more accurate result.