Joshua Donahoe

Programming Assignment 5

Binary Search Tree with Spellchecker

07/21/2015

Binary Search Tree with Spellchecker

Programming assignment five required us to implement a spellchecker using a different data structure from our previous spellchecker iterations. The program was written very similarly to the last assignment that used linked lists. The only things that was changed was what data structure we used to store the dictionary, and how we searched through the program. What the program required us to do was utilize this new data structure to spellcheck a text document that went through a parser. Each word in the file then was compared with our dictionary which was a binary search tree. We then properly incremented the proper counters based on whether it was found or not found.

This program ended up being very interesting in terms of comparing it based on efficiency with the other spellcheckers that we wrote previous. In programming assignment two we utilized a large dictionary array that was searched by a simple binary search method. It took the program roughly six seconds to execute that part of the code. In programming assignment four we implemented a less efficient model of the spellchecker. We used 26 linked lists that contained words based on their first letter, but the lists were not sorted beyond that. Then the program searched for the word using its first letter as a basis for which list to traverse. The program took roughly 28 seconds. It took a lot longer than just the simple binary search because of how many comparisons it roughly would have to make. A binary search only took about 17 comparisons to figure out a word was not found, but the linkedList required comparisons equal to the number of words in the respective linkedList for a word that wasn’t found since it wasn’t in any order. The last version of the spellchecker however ended up being the most efficient because it mixed these two ideas of separating it alphabetically and having some sort of sorted methodology to it. This program only took around four seconds to complete and only took about 10 comparisons to figure out if a word was in the dictionary or not.

These assignments just displayed that different algorithms can heavily impact run-time, and that efficiency is a very important part of software development.

Outputs:

```
Time elapsed: = 3.788401 seconds

Total words found: 939674

Total comparisons for words found: 15284398

Average comparisons per word found: 16.26563893435383

Total words not found: 52466

Total comparisons for words not found: 515894

Average comparisons per word not found: 9.83292036747608
```