Tyler Nally

11/7/15

Assignment 5

Binary Search Tree Spell Checker Program

Extended Abstract

In java, a binary search tree is a container that stores objects in memory to allow for fast searches deletions, additions, etc. We tested how well the binary search tree can accomplish the task of spellchecking a txt file. This requires random elements to be accessed in the tree, and then compared. Two different txt files were used in the program. One was oliver.txt, which is the file to be spellchecked. The second was dictionary, which is a collection of words to compare the text from oliver.txt with.

The dictionary file contained a list of words in random order. The oliver.txt file is an excerpt from The Project Gutenberg EBook of The Complete PG Works of O. W. Holmes, Sr.
The oliver.txt file was compared with the dictionary list using appropiate methods. We created 26 BSTs to store the dictionary file. Second, the dictionary file had to be read, and then copied into the correct tree alphabetically. Next, the oliver.txt file had to be read, then compared with the words in the binary search trees. To accomplish this, we had to find the correct tree to compare with. The first letter was compared, allowing the program to find the correct word within the tree. After the correct collection was located, the entire word is compared with all of the words in that collection. After the words are compared, the program decides whether the word is spelled correctly and returns true or false.

In this program, we implemented several different counters. If the word was spelled incorrectly, the “wrongwords” counter was incremented. If the word was spelled correctly, the “rightwords” counter was incremented. This allows the program to keep track of how many words are spelled correctly, and how many were not. The comparisons were also accounted for. If a word was compared or not compared, then the appropriate counter was incremented. This allows the user to see what words in the dictionary were found in the txt file. After all of the words are compared and counted, we took the number of wrong words and comparisons not found to find the average number of comparisons not found. The right words and comparisons found gave us the average number of comparisons found.

The BST’s performance was pretty good. The program executed pretty quickly but I would say that the binary search was more efficient than the linked lists. The program’s build time gave a faster output.