Indexer CS 214

Scherer, Wood

The purpose of this program is to take in two arguments the first gives the name of a file for the program to create and fill with an index of words found in the directory or file name in the second argument. The file created by the program stores a list of all the words found in alphabetical order after each word it gives all the file that word was found in in addition to how many times it occurred in each file.
The data structure: We stored the words in a Binary Search Tree sorted by alphabetical order. Then we have an unsorted Linked List that stores both file name and the number of occurrences in that file.

1) First, we check to see if the program has arguments, if so we continue to determine if the second argument is a file or directory. If it's a directory the directory is then searched through and the directoryhandler is called each time a directory is found and filehandler each time a file is found.

2) Filehandler iterates through the file using tokenizer which returns each word it comes across. After receiving a word filehandler then gives that word to sorted-list which then updates the data structures accordingly.

4) After finishing iterating through all the directories and files the program then unloads all the information into a file created using the first argument it was given. The file created by the program stores a list of all the words found in alphabetical order after each word it gives all the file that word was found in in addition to how many times it occurred in each file. To do this it goes through the binary search tree recursively left to right printing out each word then accessing the linked list stored in that words sorting it numerically and then printing the file name and occurrences.

Complexity Analysis:

Run time of each major component:
tokenizer = O(1)
SLInsert = O(log(number unique words) + number of unique files)
Explanation: Insertion into the BST and iterating through the LL.
Writetofile = O(log(number of unique word) + number of unique words * number of files ^ 2)
Explanation: Iterating through the BST and then sorting LL in numerical order
Total runtime = total number of words * tokenizer + total number of words * SLInsert + writetofile
Total runtime = total number of words + total number of words * (log(number unique words) + number of unique files) + log(number of unique word) + number of unique words * number of files ^ 2)

O(total number of words * log(number unique words) + number of unique words * number of files ^ 2)

Let n = total number of words w = number of unique words and f = number of files

Total = O(n * log(w) + wf^2)

Our Data Structure Visualized (BST with Linked Lists): Essentially, the nodes represent the structure of the binary search tree while each node acts as a linked list of any Records if they are the same “token”.