Chris LoPresti

Roslan Arslanouk

CS214

Asst 2: Keyspace Construction

This program, given a path to a file or directory, will parse the file/directory and create an inverted index. An inverted index maps each token found in a file to a filename. For example, if a file named hello.txt is found in multiple directories, all the words found in these multiple files will map to a file named hello.txt. It also maintains the frequency with which each token appears in each file. The program will tokenize the strings in the files and produce an inverted index of how many times the word occurred in each file, sorted by word (a -> z). All the words that are found will be converted to lower case before being stored. Once the program is done storing tokens, it writes to an output file in the given XML format found in the assignment description. All words are in alphabetical order a->z, however, the file names are printed in descending order of the given frequencies of each word. If two filenames have the same word, with the same frequency of that word, then the file names will be sorted a->z in that case alone. An example can be found in the testplan.txt file.

To store the tokens the program uses a Binary Tree data structure where each node corresponds to the word (token). Each node also has two pointers, to a left and right child, and a pointer to a Linked List. In the Linked List, each node contains the name of the file that contains the given word as well as its frequency. Additionally, the program has a timer that counts the time while it is running and displays this time in the command line.

When analyzing our program, we can see it takes O(nlog n) time to do a sorted insert, and O(k^2 + nlog n) to write to the output file. Our data structure is a BST. The reason we have O(k^2 + nlog n) when printing is because we need log n time to get to each level of the tree, and k^2 time to sort the list of filenames at each level, were k is the number of files containing a given word. These are the two main runtimes that affect our program. Reading files and going through directories just takes a given time depending on the number of files and directories.

For example, given the following files:

File path	File content
/dir/test	Test me please
/dir/test2.txt	test this please
/dir/nextdir/test.txt	wow

The program would output:

```
<"?xml version="1.0" encoding="UTF-8"?>
<fileIndex>
    <word text="me">
        <file name="test.txt">1</file>
    </word>
    <word text="please">
        <file name="test2.txt">1</file>
        <file name="test.txt">1</file>
    </word>
    <word text="test">
        <file name="test2.txt">1</file>
        <file name="test.txt">1</file>
     </word>
    <word text="this">
        <file name="test2.txt">1</file>
    </word>
     <word text="wow">
        <file name="test.txt">1</file>
```

```
	</word>
<fileIndex>
```