Indexer Readme

Timothy Park tjp135
Connor O'Brien jo225

Intro:
Our program took a directory/file as an input and output a file with content that included a token of each valid word (according to the definition given) and sorted by count and alphabetical order, in JSON format.

Indexer:
Our indexer project is broken down into 3 main parts, Indexer, Sorted-List, and Tokenizer.

Indexer.c recursively calls on itself to go through each directory and stops when all files/directories have been opened.
This file also contains the FileSorter() function where it sorts all the records of each given token.
As well as the JSONprint() function where it goes through each "node" and prints out in JSON format.

Tokenizer.c goes through the file and scans the content of the file and breaks it down into tokens based on the definition given, where it must start with an alpha character. ** (In our code, if it started with a number, we excluded the number until it reached an alpha character).

Sorted-list.c creates the initial list where it makes the nodes of each token and puts the records according to each token.

Space/Time analysis:

According to space, we malloc a block of memory according to the size of fscanf output.

Most of the time consists in the fscanf of each file, the sorting of each token given from that file, and sorting all the records at the end, and printing the list.

The time of fscanf of each file depends on the length of the file.

if the length of the content is n, there are AT LEAST n iterations because there is a pointer that goes through each letter to see if is valid.

the sorting of each token depends on whether there are many non alpha numeric characters, because of the continuous breakdown of each token.

the sorting of records is worst case O ($n^2 * m$) because sorting takes n^2 for each token m .

So overall it would be big O($n^2 * m$) because the other two functions are under the sorting.