Rushit Sanghrajka

Ethan Berroa

Systems Programming 214

Programming Assignment 4: Indexer

In this assignment, we implement an indexer, which simply takes a directory name. The indexer then searches for text files within the directory, and starts building an index for the tokens that it comes across, and starts indexing them. After indexing them, it writes the inverted indexes to a JSON file.

The indexer keeps a track of all the tokens it comes across, along with the source file for that particular token and the frequency that it appears with in that particular source file. It sorts these tokens lexically.

How it works:

The basic design of our program splits the Indexer into different modules. The getContents function opens the directory and recursively traverses through the directories and retrieves text files within the directories. These text files are passed to the getContentsforFile function.

The getContentsForFile function opens the file and breaks the text into tokens. Each token is now passed into the Tokens, and for each token, it creates an entry for Sources.

Tokens is a sorted list, it uses our sorted list implementation from assignment 2. Each node in this sorted list holds a token, and a pointer to the Sources list.

Sources is a sorted list, which also uses our sorted list implementation from assignment 2. Each node in this sorted list (source node) holds the path of the file, and the frequency of the token appearing in the file.

Once getContents returns the final inverted indexes, we now simple traverse through the data structure and build a JSON data file.

Efficiency:

Space efficiency:

The sorted-list implementation holds the tokens and sources, which are strings who's lengths are built dynamically. Frequency is an int value, and the number of pointers is the sum of the number of source file the token appears in and the number of unique tokens in the file.

Time Efficiency:

For building the data structure, we have an efficiency of O(m+n) approximately. It traverses through the list linearly to find the token, and once that token is found it traverses through the sources. So m in this case would be the number of tokens, and n would be the number of files the tokens appear in.

For writing the data to JSON, the Indexer would have an efficiency of O(mn). This is because it accesses every element while writing it. Once again, m is the number of tokens and n is the average number of files each token appears in.