Tiny Search Engine: Lab 5 Indexer Code Documentation

Author: Kartik Menon
Date: May 2014
Platform Tested On
```
Linux spruce.cs.dartmouth.edu 3.13.10-200.fc20.x86_64 #1 SMP Mon Apr 14
20:34:16 UTC 2014 x86_64 x86_64 x86_64 GNU/Linux
```

Design Specification

1 Input
Command line input usage:
Regular Mode:
./indexer [DIRECTORY] [RESULTS FILENAME]
Example: ./indexer ~cs50/tse/crawler/lvl0 index.dat
Testing Mode:
./indexer [DIRECTORY] [RESULTS FILENAME] [RESULTS FILENAME] [REWRITE FILE]
Example: ./indexer ~cs50/tse/crawler/lvl0 index.dat index.dat indexOut.dat

[DIRECTORY] ~cs50/tse/crawler/lvl0
Requirement: The directory and all the files contained in it are readable. The files contained in the directory are of the format:
Line 1: <Webpage URL>
Line 2: <Depth of crawl>
Line 3 to Line *n*: <HTML>
Usage: Indexer will complain if no directory is specified or if the files inside it can't be opened.

[RESULTS FILENAME] index.dat
Requirement: If index.dat already exists, it must be readable.
Usage: Indexer will complain if it isn't readable. If it already exists, Indexer will warn that it will be overwritten.

[REWRITE FILE] indexOut.dat
Requirement: If indexOut.dat already exists, it must be readable.
Usage: Indexer will complain if it isn't readable. If it already exists, Indexer will warn that it will be overwritten.

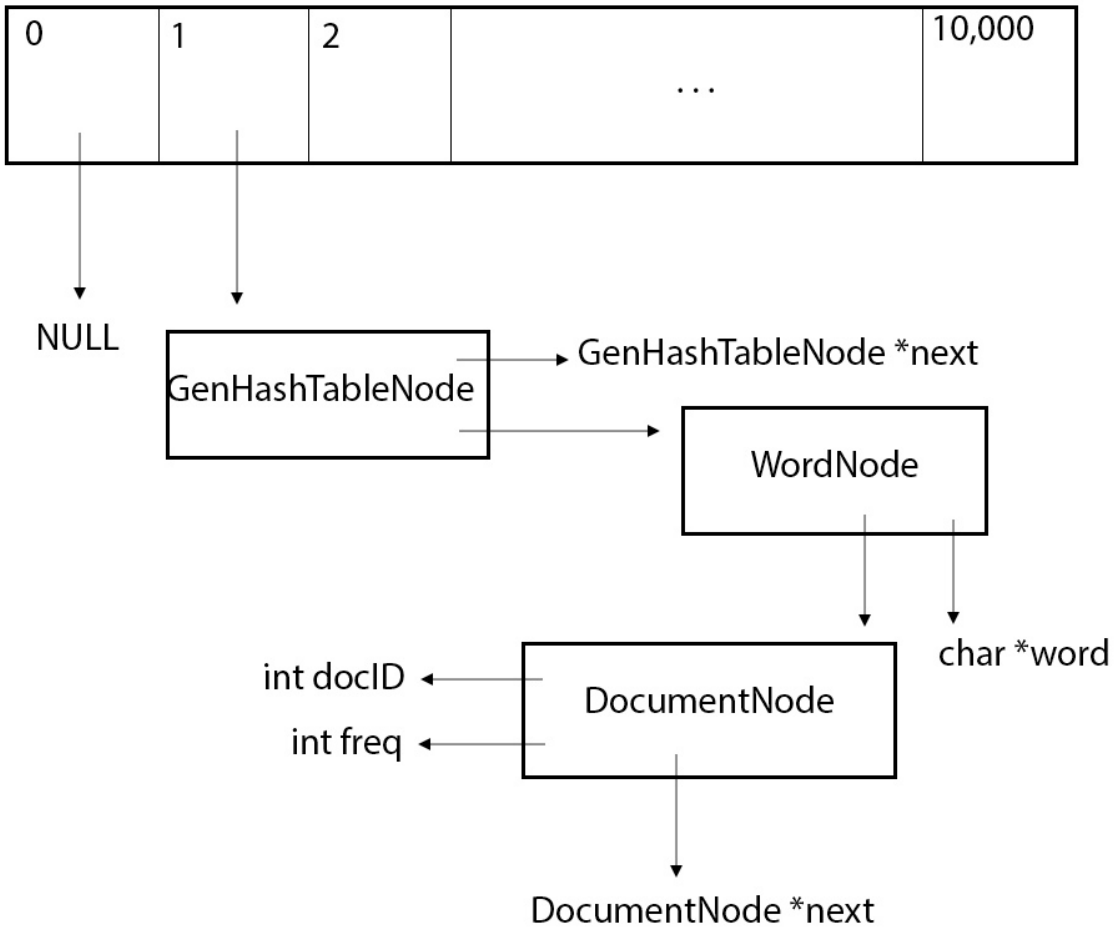
2 Output
In regular mode, indexer will produce or write to an existing file [RESULTS FILENAME] the indexed data from the files in [DIRECTORY]. The files will be of the format:
word 1 [number of documents found containing word 1] [doc ID instance of word 1] [frequency]
Example:
foop 3 124 4 11 2 43 1
The word "foop" was found in 3 documents: 4 times in document 124, 2 times in document 11 and 1 time in document 43.

3 Data Flow

a. Crawled HTML data stored into a string.
b. String parsed for words not containing HTML tags.
c. Words are slotted into hash table with appropriate WordNodes and DocumentNodes.
d. The contents of the hash table are printed out to [RESULTS FILENAME]
e. If testing mode is on, another hash table is created and the data from [RESULTS FILENAME] is slotted into it, and the hash table is printed out again for confirmation.

4 Data Structures

A HashTable of GenHashTableNodes, each with a (void*) hashKey containing a WordNode and a next pointer to another GenHashTableNode. Each WordNode has a pointer to a word character and a pointer to a DocumentNode. Each DocumentNode has a docID and freq (ints) and a pointer to another DocumentNode.



5 Pseudocode

~ Initialize a hash table to store WordNodes and DocumentNodes.
~ Get all of the files by names in the directory specified by [DIRECTORY]
~ For each file, extract the HTML and loop over the HTML until all the words have been found. Throw out the word if it is less than three characters.
~ Update the hash table index with the word and new document ID.
    - If nothing exists in the spot hashed to by JenkinsHash(word), create a new WordNode for the word and a new DocumentNode for the document ID and store them in the hash table.
    - Otherwise, for all of the nodes in the linked list of HashTableNodes in each has bucket:
        - If a WordNode corresponding to the word is found, go through the DocumentNodes.
        - If a DocumentNode for the document ID is found, increment frequency.
        - If not, create a new DocumentNode with frequency = 1.
    - If those conditions fail, the new word is colliding with an existing word in the hash slot. Create a new WordNode and DocumentNode, sticking them into a new HashTableNode, and then sticking the new HashTableNode into the hash table.
~ When the hash table is updated, loop over all of its data and print them formatted to the file specified by [RESULTS FILENAME]

~ If testing mode is on, reload the hash table. Essentially the same method as updating the hash table index, read in all the data from the [RESULTS FILENAME] file and put it into the appropriate place in the hash table.
~ Call the same function to save the reloaded hash table data into [REWRITE FILE].
~ Free all data.

6 Error Conditions

~ Indexer has a function to validate arguments that checks if the number of arguments is not 3 or 5 (including the executable call) and checks that the directory exists. It also checks to see if the files that are given to indexer exist, and if they do indexer warns the user but does not exit.
~ Everytime a file is opened or something is allocated in memory, a check is performed.
~ Several checks are performed when loading the HTML from the file. If the file only has two lines (i.e., a URL and a depth) or if the file is empty, indexer will print to stderr and exit.
~ If a file is improperly named (like not 1, 2, … n), indexer will see that and exit.
~ If a directory is empty, nonexistent, or unreadable (i.e., chmod –r dirName) indexer will exit.

7 Files Used

- Web.[ch] for parsing HTML for words.
- header.h for defining WordNode and DocumentNode structs and MALLOC_CHECK macro.
- indexer.c
- file.[ch] for getting files in a directory.