Tiny Search Engine: Lab 6 Query Code Documentation

Author: Kartik Menon
Date: May 2014
Platform Tested On
```
Linux wildcat.cs.dartmouth.edu 3.14.2-200.fc20.x86_64 #1 SMP Mon Apr 28
14:40:57 UTC 2014 x86_64 x86_64 x86_64 GNU/Linux
```

Design Specification

1 Input
Command line input usage:
./queryEngine [INDEXER DATA FILE] [CRAWLED DATA DIRECTORY]
Example: ./queryEngine ~cs50/tse/indexer/cs_lvl3.dat ~cs50/tse/crawler/lvl3

[INDEXER DATA FILE] ~cs50/tse/indexer/cs_lvl3.dat
Requirements: The file must be openable and the format of the file must be as follows:
<Word> <Total Occurences> <docID 1> <freq 1> … <docID *n*> <freq *n*>

Usage: Query engine will complain and quit if the indexer file can't be opened.

[CRAWLED DATA DIRECTORY] ~cs50/tse/crawler/lvl3
Requirements: The directory must have numbered files (starting at 1) each representing a different crawled website. The first line of the file must be the URL of the website.

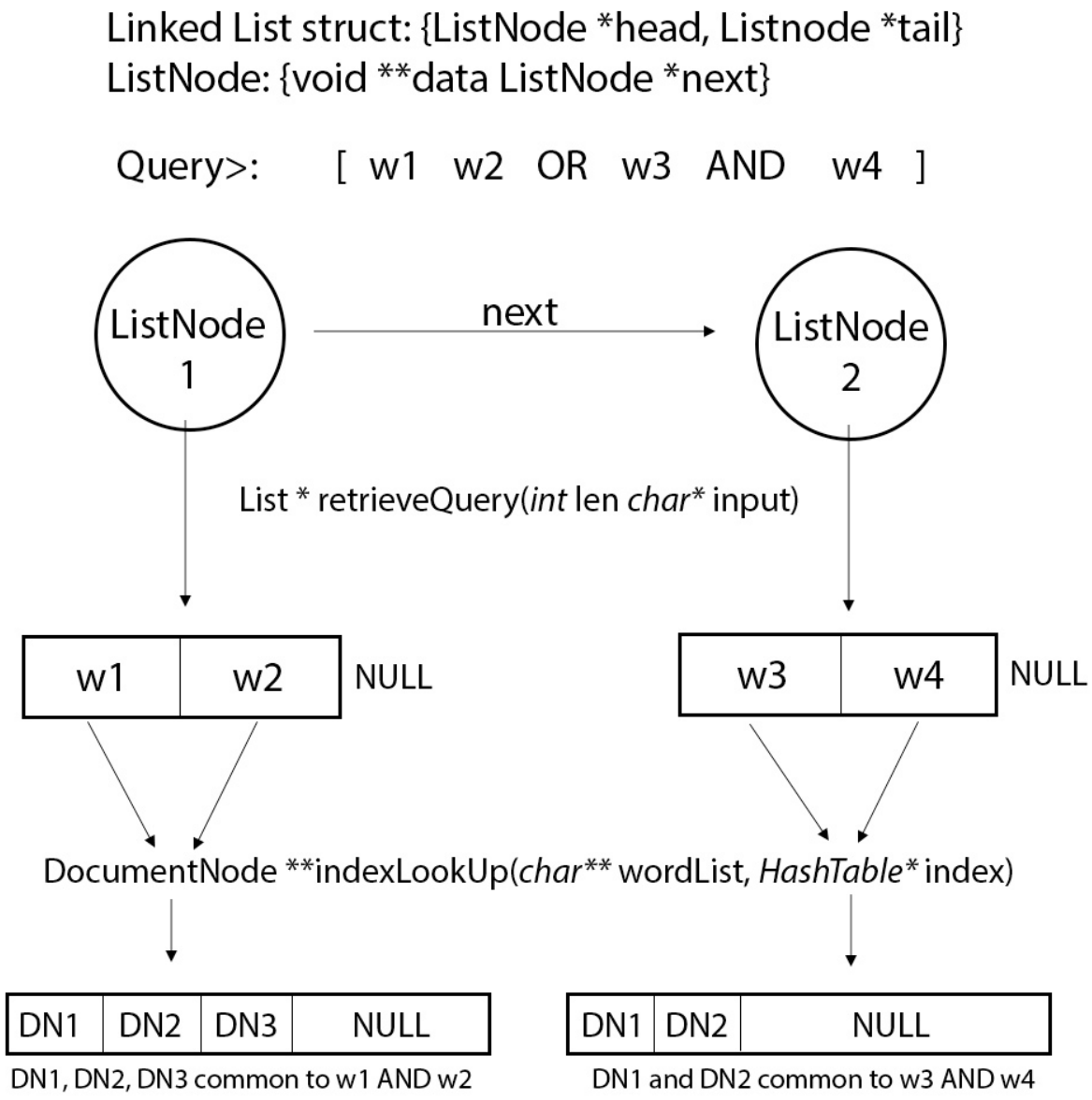
2 Output
When the query engine has reloaded the hash index from the indexed data file, it will print a prompt: QUERY:> where the user can type in query requests. Inputted words will be looked up in the hash table, sorted according to their frequency in the crawled data documents, and returned to the user. If multiple words are inputted, the query engine will find the intersection of all those words (i.e., only return documents that contain all of the queried words). Words are separated by "AND," have the same output. Intersected words are sorted by a combined frequency of all of the relevant words. If words are separated by an "OR," all of the document information associated with the words on either side of "OR" will be returned. Words common to a document will be ranked according to the sum of their frequencies.
Example:

```
QUERY:> dog
DocumentID:73 URL:http://old-
www.cs.dartmouth.edu/~cs50/tse/wiki/Legged_Squad_Support_System.html
DocumentID:216 URL:http://old-
www.cs.dartmouth.edu/~cs50/tse/wiki/Parsing.html
DocumentID:25 URL:http://old-
www.cs.dartmouth.edu/~cs50/tse/wiki/Ada_Lovelace.html
DocumentID:35 URL:http://old-
www.cs.dartmouth.edu/~cs50/tse/wiki/Enigma_machine.html
DocumentID:46 URL:http://old-
www.cs.dartmouth.edu/~cs50/tse/wiki/Human_Genome_Project.html
```

```
DocumentID:59 URL:http://old-
www.cs.dartmouth.edu/~cs50/tse/wiki/Entertainment.html
```

If when using "AND" operators one of the words is not found in any of the documents, the query will fail.

3 Data Flow



This is how the query engine processes queries. Explanation follows on the next page.

1. Collect the query by *fgets()*, which will break if the user inputs a EOF (=^D)
2. The query is broken up by strtok(), unless the word is an "AND" or an "OR", as these are operators, not queries.
3. Create a linked list, with each ListNode pointing to an array of characters. This array will store query requests.
4. If the connector between words is a space " " or an "AND", add the word to the next index in the same ListNode's array. If the connector is an "OR", make a new ListNode and start a new array of characters, putting the word following the "OR" inside there.
5. When *retrieveQuery()* is finished, have a linked list of char arrays, filled with queries, with nodes separated by "OR" operators.
6. For each ListNode in the list, pass the ListNode's data (a char array) to *indexLookUp()*, which will look up all of the document nodes for all of the words in the char array. If there are multiple words in the array, (i.e., the user used an "AND" or " " operator) *indexLookUp()* will also find the intersection of all of the DocumentNodes for each word, increasing the ranking of each query word for intersections.
7. *indexLookUp()* returns a pointer array filled with DocumentNodes that are common to all of the words supplied to it in the char array.

Then "OR" operators are taken care of by *combineArrays()*. In the main loop I keep two pointer arrays of DocumentNodes; one to hold the most recently returned array from *indexLookUp()*, which intersects and returns the DocumentNodes for a single ListNode, and one to hold the total DocumentNodes to return to the user. If an "OR" is used, add on the DocumentNodes of the word following the "OR" to the combined, or total, array of DocumentNodes. This is done in *combineArrays()*. It looks for DocumentNodes that are common to the total array and the newest array, and if there are common elements, update the frequency of the DocumentNode to the sum of the two, and if there aren't common elements, just add on the element to the end of the total array.

Next is *rankQuery()*, which is a simple implementation of an array sort, which sorts on the frequencies of the DocumentNodes in the combined DocumentNode array. This returns another DocumentNode array *ranked*, which puts the DocumentNodes in the right order. This *ranked* array is then printed out in order to the user, and the loop repeats, allowing for multiple queries.

4 Pseudocode

- Until the user enters an EOF (=^D) character, read the line in, up to 1000 characters (1001 including ending null character)
- Store each subsequent word (separated by a space) and words separated by the "AND" operand inside a char * pointer array, which is stored in a ListNode of a linked list. If an "OR" is seen, create a new linked list node and start a new char * pointer array in there. See diagram above.
- Loop over all of the linked list nodes, and for each of them take the intersection of all the document nodes in each node's array of words; this covers the "AND" operand (equivalent to the " " operand).
- Add the relevant document nodes to a combined list. Keep looping; if multiple nodes are in the list, an "OR" was part of the search query; make sure that none of the "OR" word document nodes are duplicated in the combined list and add them onto the end, reallocating space in the combined list for them.
- Once all of the relevant document nodes have been added to the combined list, rank them in order of their frequency. In cases of both "AND" and "OR" operands, the frequency for commonly occuring nodes is added together. Ranking is a simple list sort based on frequency.

- Locate the correct URL from the newly made ranked array of document nodes and print them to the user.
- Free resources.

5 Error Conditions

- Validate arguments – if not three then complain.
- Check that the indexer data file supplied to the query engine is readable.
- Check that the crawled data directory supplied to the query engine is readable.
- Check that the user has given valid input (nothing like just a newline or non alphabetical characters, for example)
- All memory allocations checked.
- Checks if the assigned word doesn't show up in the hash table (i.e., no results found)
- Will not take (or rather, separates) queries that are more than 1000 characters (or 1001 including the null-terminating byte)

6 Files Used

- query.c
- reload.[ch]
- util library