Search Engine Indexer Documentation

Delos Chang

Design Specification:

- **Assumptions**
  - The HTML files are stored as numbers sequentially. Example: "1", "2", "3". The indexer will check for this format and avoid directories / non-numbered names
  - The indexer *will overwrite* existing files when it writes out the index.
- **Functional Decomposition:**
  - Indexer.c
    - **3-argument:** Extracts all keywords for each stored webpage and records the URL where each word was found. A lookup table is created that maintains the words that were found and their associated URLs. The result is a table that maps a word to all the URLS and the word frequency, where the word was found.
    - **5-argument:** Creates index from the files and writes out like the 3 argument indexer. In addition, it will "reload" the index file (4^{th} argument), load it back into memory, and write it out to a new index file (5^{th} argument). BATS.sh will then test the integrity of the files using "diff"
  - Indexer.h
    - Header file for indexer. Contains useful constants and macros like BZERO and MALLOC_CHECK. Includes data structures and prototypes (mentioned later).
  - File.c
    - Scans through the target directory (from arguments parameters) and returns the number of files within that directory.
  - File.h
    - Header file for file.c. Includes prototypes.
  - Hash.c
    - Provides a hash function so the indexer can look up the existence of a WordNode quicker.
  - BATS.sh
    - A script that tests the invalid parameters. Also checks if the indexed file and reloaded index file are the same using the command "diff"
    - Also outputs everything from stdout and stderr into a log file with a datetime stamp
- I/O
  - **3-argument:** ./indexer [TARGET_DIRECTORY] [RESULTS FILENAME]

        - **e.g.** ./indexer ../crawler_dir/data index.dat
        - **output:** a file named [RESULTS FILENAME] with all the parsed words from [TARGET_DIRECTORY]
    - **5-argument:** ./indexer [TARGET_DIRECTORY] [RESULTS FILENAME] [RESULTS FILENAME] [REWRITTEN FILENAME]
        - **e.g.** ./indexer ../crawler_dir/data index.dat index.dat new_index.dat
        - **output:** a file named [RESULTS FILENAME] with all the parsed words from [TARGET_DIRECTORY] **and** another file that is the result of "reloading" the index back to memory and writing it out again at [REWRITTEN FILENAME]
- **Major Data Structures**
    - Inverted Index
        - This index holds hash slots that point to the first WordNode for faster indexing. For example, if the word is 'dog' and the hash function computes 19. Hash[19] will point to the first WordNode that is in that slot.
    - WordNode
        - This data structure is a doubly linked list with no sentinel. It is used to handle collisions at a hash slot. For example, if more than one word has the same hash number from the hash function, it will be added to the end of the WordNode list. This design decision allows faster searching because given a word, only a cluster needs to be evaluated (hash function computes at O(1)).
        - The WordNode also holds a pointer to the first element of a DocumentNode. All DocumentNodes in *that* list has the word.
    - DocumentNode
        - This data structure holds the documentID specifier and is a linked list with only a next pointer. It also holds the frequency that the word (in the WordNode this DocumentNode is pointing to) has occurred.
- **PsuedoCode**
    - 1. Program parameter processing (make sure valid args)
    - 2. Initialize the inverted index
    - 3. Scan the target directory for number of files
    - 4. Loop through each file and extract the words. Filter the newlines and generally non-alpha characters.
    - 5. Loop through each of the extracted words and update the inverted index.
        - 5b. Hash the word. Check the hash slot.
            - 5b(i) If the slot the word is hashed to is occupied already, loop through each of the WordNodes and check if it exists already.
                - If it does exist, check if the DocumentNode exists.

        - If the DocumentNode exists, increment the page frequency count.
        - If not, create a new Document Node and add it to the end of list. Initialize its page freq to 1.
    - 5b(ii) If the slot is not occupied, create a new WordNode and DocumentNode and insert it in.
- Loop through each of the hash slots, for each, loop through the WordNode. Write the word and the number of DocNodes it has to a file.
    - Loop through the DocNodes and write the Document ID and the page word frequency into [RESULTS FILENAME]
- Clean up the resources by freeing memory and closing files
- Check if there are 5 arguments. If so, do the additional following:
    - Reload the index from the file by reading each character from the file in.
    - Write the file back out
    - Test using "diff" to make sure both the original and "reloaded" file are the same

Functional Specification:

- **3-argument mode:** Load the document from file system.
- Use document filename as the unique document specifier
- Parse the document discarding the HTML tags in order to count occurrences of each word fount in the document
- Store the word and frequency of occurrence in a document in an inverted index data structure so information can be easily retrieved.
- After all documents are processed, indexer should save the information to a file. This file will be used by the query engine.
- Recreate the index: write the index out to a file and store it
- **5-argument mode:** Read the index in from the file and recreate the index in another copy
- Write out the index to a different file and compare it with the original using "diff". There should be no difference.

Summary of error conditions detected and reported:

- Insufficient or incorrect arguments passed in (stops indexer).
    - Incorrect number of arguments
    - Nonexistent or unreadable [TARGET DIRECTORY]
- No files in the TARGET_DIRECTORY (stops indexer)
- Files in the [TARGET_DIRECTORY] are not formatted correctly
    - e.g. "abc" "def" "1 (folder)" would not be read in by the indexer. (stops indexer – crash early and often)
- Malloc out of memory or file unable to open (via fopen)
- File already exists for [RESULTS_FILENAME] (warning)
- File already exists for [NEW_FILENAME], 5th argument (warning)

- System commands do not have an exit status of 0 (stops indexer)
- Filesize is invalid for reading file into buffer (via ftell) (stops indexer)
- Error reading the file into buffer (via ftell) (stops indexer)
- Unsuccessfully indexing a word via updateIndex (warning)
- Unsuccessfully reconstructing the index froma file (warning)
- Unable to initialize data structures (stops indexer)
- Unable to save the index into file (stops indexer)
- Unable to reload the index frlom the file (stops indexer)

Test cases, expected results

- Insufficient arguments – program will print usage information and report the number of arguments (need either 3 or 5)
- Invalid or unreadable directory – program will print usage information and ask for an existent/valid directory
- No files in the target directory tested – stops indexer,
- Wrong format of file in the target directory – indexer skips over the offending file/folder and does not include it in the count
- File in the "results filename" already exists – indexer warns but does not stop
- File in the target LOAD_FILENAME for reindexing doesn't exist – indexer stops and prints usage information and asks for an existent file
- Target directory with a trailing slash and no trailing slash – indexer works witih either (either are valid)
- Checking whether the "reloaded" index file and original are the same – if not, indexer will tell you the result from the BATS.sh
- Not enough memory to malloc – indexer will stop and tell user that they are out of memory
- Invalid files to open (file pointer comes up null) – indexer will stop and tell user that the file was unable to be opened.