Programming Assignment 4

Little Search Engine

In this assignment you will implement a simple search engine for text documents using hash tables.

Worth 80 points = 8% of your course grade

Posted Fri, Mar 29

Due Tue, Apr 16, 11:00 PM (WARNING!! NO GRACE PERIOD)

Extended deadline (with ONE time free extension pass): Fri, Apr 19, 11:00 PM (NO GRACE PERIOD)

You get ONE free extension pass for assignments during the semester, no questions asked. There will be a total of 5 assignments this semester, and you may use this one free extension pass for any of the 5 assignments.

A separate Sakai assignment will be opened for extensions AFTER the deadline for the regular submission has passed. The extension will be 3 days (72 hours). If/when you choose to use the one-time extension, you don't need to ask for permission - just drop your submission in the extension assignment. And you can do this even if you dropped something in the regular submission, and later on decided to use the extension - in this case, only the extension submission will be graded, the regular submission will be ignored.

- You will work on this assignment individually. Read DCS Academic Integrity Policy for Programming Assignments - you are responsible for abiding by the policy. In particular, note that "**All Violations of the Academic Integrity Policy will be reported by the instructor to the appropriate Dean**".
- **IMPORTANT - READ THE FOLLOWING CAREFULLY!!!**

  Assignments emailed to the instructor or TAs will be ignored--they will NOT be accepted for grading. We will only grade submissions in Sakai.

  If your program does not compile, you will not get any credit.

  Most compilation errors occur for two reasons:

  1. You are programming outside Eclipse, and you delete the "package" statement at the top of the file. If you do this, you are changing the program structure, and it will not compile when we test it.
  2. You make some last minute changes, and submit without compiling.

  To avoid these issues, (a) START EARLY, and give yourself plenty of time to work through the assignment, and (b) Submit a version well before the deadline so there is at least something in Sakai for us to grade. And you can keep submitting later versions (up to 10) - we will accept the LATEST version.

Summary

You will implement a little search engine to do two things: (a) gather and index keywords that appear in a set of plain text documents, and (b) search for user-input keywords against the index and return a list of matching documents in which these keywords occur.

Implementation

Download the attached `lse_project.zip` file to your computer. DO NOT unzip it. Instead, follow the instructions on the Eclipse page under the section "Importing a Zipped Project into Eclipse" to get the entire project, called `Little Search Engine`, into your Eclipse workspace.

Here are the contents of the project:

- A class, `lse.LittleSearchEngine`. This is where you will fill in your code, details follow.
- A supporting class, `lse.Occurrence`, which you will NOT change.
- Two sample text documents, `AliceCh1.txt`, and `WowCh1.txt`, directly under the project folder, for preliminary testing. Be sure to get other online text documents--or make your own--for more rigorous testing.
- A `noisewords.txt` file that contains a list of "noise" words, one per line. Noise words are commonplace words (such as "the") that must be ignored by the search engine. You will use this file (and this file ONLY) to filter out noise words from the documents you read, when gathering keywords.
- A `docs.txt` file that has a list of all documents (in this case `AliceCh1.txt` and `WowCh1.txt`) from which the search engine should extract keywords.

NOTE: You will need to write your own driver to test your implementation. This driver can take as inputs a file that contains the names of all the documents (such as `docs.txt`), as well as the `noisewords.txt` file. It can then set up a `LittleSearchEngine` object and call its methods as needed to test the implementation. The `docs.txt` and `noisewords.txt` filenames will be sent in as the arguments to the `makeIndex` method in `LittleSearchEngine`.

Following is the sequence of method calls that will be performed on a `LittleSearchEngine` object, to index and search keywords.

- `LittleSearchEngine()` - Already implemented.

  The constructor creates new (empty) `keywordsIndex` and `noiseWords` hash tables. The `keywordsIndex` hash table is the MASTER hash table, which indexes all keywords from all input documents. The `noiseWords` hash table stores all the noise words. Both of these are fields in the `LittleSearchEngine` class.

  Every key in the `keywordsIndex` hash table is a keyword. The associated value for a keyword is an array list of (document,frequency) pairs for the documents in which the keyword occurs, *arranged in descending order of frequencies*. A (document,frequency) pair is held in an `Occurrence` object. The `Occurrence` class is defined in the `LittleSearchEngine.java` file, at the top. In an `Occurrence` object, the `document` field is the name of the document, which is basically the file name, e.g. AliceCh1.txt.

- `void makeIndex(String docsFile, String noiseWordsFile)` - Already implemented.

  Indexes all the keywords in all the input documents. See the method documentation and body in the `LittleSearchEngine.java` file for details.

  If you want to index the given sample documents, the first parameter would be the file `docs.txt` and the second parameter would be the noise words file, `noisewords.txt`

  After this method finishes executing, the full index of all keywords found in all input documents will be in the `keywordsIndex` hash table.

The `makeIndex` methods calls methods `loadKeywordsFromDocument` and `mergeKeywords`, both of which you need to implement.

- `HashMap<String,Occurrence> loadKeywordsFromDocument(String docFile)` - You implement.

  This method creates a hash table for all keywords in a single given document. See the method documentation for details.

  This method MUST call the `getKeyword` method, which you need to implement.

  - `String getKeyword(String word)` - You implement.

    Given an input word read from a document, it checks if the word is a keyword, and returns the keyword equivalent if it is.

    FIRST, see the method documentation in the code for details, including a specific short list of punctuations to consider for filtering out. THEN