CSCI-4430/6430 Programming Languages Fall 2014
Programming Assignment #3

*This assignment is to be done either **individually** or **in pairs**. **Do not show your code to any other group** and **do not look at any other group's code**. Do not put your code in a public directory or otherwise make it public. However, you may get help from the TAs or the instructor. You are encouraged to use the LMS Discussions page to post problems so that other students can also answer/see the answers.*

Search Engine Index Builder

Given a collection of documents, a search engine indexes the terms in these documents and responds to queries consisting of terms with a ranked list of documents.

For the sake of ranking documents, the importance of each term needs to be measured quantitatively, with regard to the document it resides in as well as the whole collection.

For this assignment, we use term frequency-inverse document frequency (tf-idf) as the importance measurement, which is defined as follows:

$$\mathrm{tfidf}(t, d, D) = \mathrm{tf}(t, d) \times \mathrm{idf}(t, D)$$

, where t is the term, d is the document, and D is the collection.

$tf(t,d)$ is defined as:

$$\mathrm{tf}(t, d) = \begin{cases} 1 + \log \mathrm{f}(t, d) & \text{if } \mathrm{f}(t, d) \neq 0 \\ 0 & \text{otherwise} \end{cases}$$

, where $f(t,d)$ is the frequency, or the number of times term t appears in document d.

$idf(t,D)$ is defined as:

$$\mathrm{idf}(t, D) = \log \frac{N + 2}{1 + |\{d \in D : t \in d\}|}$$

, where N is the total number of documents in the collection D, and $|\{d \in D : t \in d\}|$ is the number of documents where the term t appears.

Note that base 10 logarithm is used throughout this assignment, and we use only words for terms.

Part 1 (70%). Concurrent programming.

You are given a directory compressed as `collection.zip` containing `N` documents, named `0.txt`, `1.txt`, ..., `<N-1>.txt`. All the documents are preprocessed so each of them contains only a sequence of lowercase ASCII terms separated with single whitespaces and line breaks.

Your task is to calculate the tf-idf value for each distinct term. The result should be 26 files in a relative path

"index", named "a.txt", "b.txt", ..., "z.txt", where each file name indicates the common initial letter of the terms in the file. e.g. "a.txt" contains all the terms starting with the letter "a". Within each file, terms should be sorted alphabetically, one term per line, in the form of:

```
[term] [doc1] [tf-idf1] [doc2] [tf-idf2] ...
```

where [term] is the lowercase ASCII term, and [tf-idf$_i$] is the tf-idf value of the term with regard to [doc$_i$], the number of a document where the term appears. e.g. "132" for 132.txt. The list of document-value pairs should be sorted according to the document number in ascending order. Only documents whose tf-idf is greater than zero must be present. All tf-idf values should be stored with exactly four decimal places (e.g., 0.0001).

Please name your program `Index.salsa` in a `pa3` module so that it can be run in the following manner.

```
$ [unzip collection.zip to the "collection" directory]
$ salsac pa3/*
$ salsa pa3.Index collection index [N] [M]
```

, where `collection` is the directory where you find 0.txt, 1.txt, ..., `index` is the name of the directory where to store index files (should be created if non-existent), [N] is the total number of files to index, and [M] is the number of actors to use to build the index.

Part 2 (30%). Distributed programming.

Write a distributed version of the program in SALSA based on Part 1. That is, you should use multiple theaters and distribute worker actors on each theater. In addition to the arguments used in Part 1, your program must accept another argument to specify a name server and theaters as follows:

```
$ salsa pa3.DistributedIndex collection index [N] [M] theaters.txt
```

The `theaters.txt` is a text file, the first line of which is the Internet address and port number of the name server and the rest is the Internet address and port number of the theaters. An example of `theaters.txt` is here.

Time-Saving Hints

0. For reference, please see the SALSA webpage, including its FAQ. Read the tutorial and a comprehensive example illustrating distributed programming in SALSA.
1. `salsac` and `salsa` are UNIX aliases or Windows batch scripts that run `java` and `javac` with the expected arguments: See .cshrc for UNIX, and salsac.bat salsa.bat for Windows.
2. To run the distributed program, first, run the name server and the theaters:

   ```
   [host0:dir0]$ wwcns [port number 0]
   [host1:dir1]$ wwctheater [port number 1]
   [host2:dir2]$ wwctheater [port number 2]
   ...
   ```

   where `wwcns` and `wwctheater` are UNIX aliases or Windows batch scripts: See .cshrc for UNIX, and wwcns.bat wwctheater.bat for Windows. Make sure that the theaters are run where the actor behavior code is available, that is, the `pa3` directory should be visible in directories: `host1:dir1` and

`host2:dir2`. Then, run the distributed program as mentioned above.

3. The theaters all cache actor behaviors. Restart all the theaters each time changes are made to the code.
4. The module/behavior names in SALSA must match the directory/file hierarchical structure in the file system. e.g., the `Index` behavior should be in a relative path `pa3/index.salsa`, and should start with the line `module pa3;`.
5. Messaging is asynchronous. `m1(...);m2(...);` does not imply `m1` occurs before `m2`.
6. Notice that in the code `m(...)@n(...);`, `n` is processed after m is executed, but not necessarily after messages sent *inside* `m` are executed. For example, if inside `m`, messages `m1` and `m2` are sent, in general, `n` could happen before `m1` and `m2`.
7. (Named) tokens **can only be used** as arguments to messages.

Extra Credit (up to 25% bonus).

1. Analyze the performance changes with different numbers of documents, actors, and machines.
2. Add a load balancing capability to your program. Instead of creating the same number of worker actors on each theater, create all workers on a single theater. Then, have other theaters steal some worker actors from the theater with many worker actors. Include the test code and an explanation of the results in your submission.
3. Implement the query answering module of the search engine. Write a search engine based on the 26 index files you have:

   ```
   $ salsa pa3.Search index [whitespace separated distinct query terms]
   ```

   where `index` is where you have your index files. Print the numbers of the top 3 documents matching the query. The matching score of a document *d* with respect to [whitespace separated distinct query terms] $t_1, t_2, \ldots t_n$ is defined as the sum of tf-idf values of all the terms with regard to the document:

   $$\mathrm{match}(d, Q, D) = \sum_{t \in Q} \mathrm{tfidf}(t, d, D)$$

   , where Q is the set of query terms.

Due Date:

Received Time	Grade Modification
before Sunday, 11/23, 11:59PM	+10%
before Monday, 11/24, 11:59PM	no modification (on time)
before Tuesday, 11/25, 11:59PM	-10%
before Thursday, 11/27, 11:59PM	-25%
after Thursday, 11/27, 11:59PM	not accepted

Grading: The assignment will be graded mostly on correctness, but code clarity / readability will also be a factor (comment, comment, comment!). See the professor or TAs, if you have ideas for other extensions for this assignment and would like extra credit for implementing them.

Submission Requirements: Please submit a ZIP file with your code, including a README file. In the README file, place the names of each group member (up to two). Your README file should also have a list of specific features / bugs in your solution. Your ZIP file should be named with your LMS user name(s) as the filename, either userid1.zip or userid1_userid2.zip. Only submit one assignment per pair via LMS.