CS533 Information retrieval

Project Part 1

Due May 3, 2011 in drop box of project

The goal of this part is to read movie reviews, and store on disk an index and a list of (document number, title) pairs.

The indexing program will read the files stored in reviews*.zip* on Blackboard and create a positional index. Since the files of the corpus are HTML files, you will need to remove tags, and “noise”. The term “noise” is used for the parts of the pages (some of the header and footer) that should not be indexed. To determine the “noise: you will need heuristics.

The index will include a dictionary of all the indexed terms, and the postings lists for each. Each record of the dictionary will contain, the term, the number of documents containing the term (df), and the offset to the posting list.

Each record in the posting list will include a docID and a position for each of two zones. The zones are: TEXT, and TITLE (of movie). The TITLE words are those contained within the TITLE tags. A record will be generated for term t if it is mentioned anywhere in the non noise part of the review.

For indexing all the non-noise words:

1. Convert all upper case letters to lower case.
2. Do not index links.
3. Do not index numbers.
4. Do index words that contain punctuations. So data-base, u.s.a, 5/12/1941, and don’t will all be index terms. A word followed by a comma, period, exclamation mark, quotes, or in parenthesis should be indexed without the punctuation.

In addition to the index you will also generate a document table with the docIDs and the title of each document, this table will be used by the retrieval program to display the results.

The input parameter to the program is:

A directory path name

The output files will be:

1. Dictionary.txt
2. Postings.txt
3. DocsTable.txt

EXAMPLE

There are 2 documents 1 and 2.

Doc 1 has the words “Titanic” in position 1 inside the title tags, and the words “boat” in position 1, “boat” in position 2, “Titanic” in position 3, “Titanic” in position 4, and “Titanic” in position 5 in the non noise part of the document.

Doc 2 has the word “Casablanca” in position 1 of the title, and the words “boat” in position 1, “German” in position 2 “German” in position 3 and “Casablanca” in position 4 of the non noise part of the document.

The doc table:

|  |  |
| --- | --- |
| Doc number | Title |
| 1 | titanic |
| 2 | casablanca |

The dictionary:

|  |  |  |
| --- | --- | --- |
| Term | df | offset |
| boat | 2 | 0 |
| casablanca | 1 | 2 |
| german | 1 | 3 |
| titanic | 1 | 4 |

The posting file

|  |  |  |
| --- | --- | --- |
| DocID | Zone 1 positions | Zone 2 positions |
| 1 |  | 1, 2 |
| 2 |  | 1 |
| 2 | 1 | 4 |
| 2 |  | 2, 3 |
| 1 | 1 | 3, 4, 5 |

**Submission directions**

You may write the code using C, C++, or Java. **Your program should compile on Bingsuns. No exceptions.** It should be purely a command line program. NO GUI will be accepted. This enables the TAs to run your code using test scripts. DO NOT assume that since your program compiled and ran correctly on your laptop it will also compile and run correctly on Bingsuns.

Please drop a *.tar.gz* file to the digital drop box of the assignment in blackboard.

The zip file should be named (lower case) as follows:

<last name>\_<first name>

When the file is unzipped it should contain a directory with the same name as the zip file.

The directory should contain the following files:

File(s) with the source code for the program and possibly a make file.

A read-me file named *readme.txt* which should contain:

Line 1: C, or C++, or Java

Line 2: Either a comma-delimited list of files, which will be compiled and linked with the appropriate compiler (gcc, g++, or javac) and then executed, or the single word “make” which will execute the makefile in the unzipped directory.  The makefile will need to produce an executable called “submission” or a “Submission” class in the java case, which will be interpreted/executed. For Java make sure that the first file in the list has the main method.

**Plagiarism Policy**

All your code will be checked for similarity to other submissions using Moss. Programmers have an uncanny knack of reproducing the same code that they have seen before. So you are advised not to look at each other's code.