CSE 435/535 Information Retrieval
Fall 2015

Programming Assignment: Boolean Query Processing based on Postings Lists

Due Date: October 19

Overview

In this programming assignment, you will be given posting lists generated from the RCV1 news corpus (http://www.daviddlewis.com/resources/testcollections/rcv1/). You need to get familiar with the data format of the given posting lists and rebuild the index after reading in the data. **Linked List** should be used to store the index data in memory as the examples shown in the textbook. **You need to construct two index with two different ordering strategies**: with one strategy, the posting of each term should be ordered by increasing document IDs; with the other strategy, the postings of each term should be ordered by decreasing term frequencies. After that, you are required to implement modules that return documents based on *term-at-a-time* with the postings list ordered by term frequencies, and *document-at-a-time* with the postings list ordered by doc IDs for a set of queries. **You should use Java for this assignment.**

Input Data format

The given postings lists can be downloaded from Piazza. The index is contained in file term.idx. The data format is summarized in the following:

- The file is a self-contained index including the dictionary. Note there is no document dictionary.
- In the index file, Each posting is on a separate line in the file
- Every posting has three values: the term, the size of the posting list and the posting list itself. Each posting is of the form X\cY\mZ where X is the term, Y is the size of posting list and Z is the posting list
- The posting list itself is expressed as [a/b, c/d, e/f,...]. The square brackets denote the start and end of the list. Each entry of the form x/y means the term occurs 'y' times in document id 'x'.

Detailed Requirements

The following functions should be implemented for this programming assignment. The results should be written into a log file (more descriptions on this latter), and the corresponding output formats are also defined in the following.

- **getTopK** *K*: This returns the key dictionary terms that have the *K* largest postings lists. The result is expected to be an ordered string in the descending order of result postings, i.e., largest in the first position, and so on. The output should be formatted as follows (K=10 for an example)

  FUNCTION: getTopK *10*
  Result: term1, term2, term3…, term10 (list the terms)

- **getPostings** *query_term*: Retrieve the postings list for the given query. Since we have N input query terms, this function should be executed N times, and output the postings for each term from both two different ordered postings list. The corresponding posting list should be displayed in the following format:

  FUNCTION: getPostings query_term
  Ordered by doc IDs: 100, 200, 300… (list the document IDs ordered by increasing document IDs)
  Ordered by TF: 300, 100, 200… (list the document IDs ordered by decreasing term frequencies)

Should display “term not found” if it is not in the index.

- **termAtATimeQueryAnd** *query_term1, ..., query_termN*: This emulates an evaluation of a multi-term Boolean AND query on the index with term-at-a-time query. Note here the number of query terms could be varied. The index ordered by decreasing term frequencies should be used in this query. Although Java has many very powerful methods that can do intersections of sets very efficiently, they are NOT allowed in this assignment. In order to learn the essence of the term-at-a-time strategy, we have to compare every docID in one posting with every docID in the other while performing the intersection. In the output file, you should display how may documents are found, how many comparisons are made during this query and how much time it takes. The document IDs should be sorted and listed.

  For example:

  ```
  FUNCTION: termAtATimeQueryAnd query_term1, …, query_termN
  xx documents are found
  yy comparisions are made
  zz seconds are used
  Result: 100, 200, 300 … (list the document IDs, re-ordered by docIDs)
  ```

  Should display “terms not found” if it is not in the index.

- **termAtATimeQueryOr** *query_term1, ..., query_termN*: This emulates an evaluation of a multi-term Boolean OR query on the index with term-at-a-time query. The index ordered by decreasing term frequencies should be used in this query. Output format is the same.

  For example:

  ```
  FUNCTION: termAtATimeQueryOr query_term1, …, query_termN
  xx documents are found
  yy comparisions are made
  zz seconds are used
  Result: 300, 100, 200… (list the document IDs, re-ordered by docIDs)
  ```

  Should display “terms not found” if it is not in the index.

- **docAtATimeQueryAnd** *query_term1, ..., query_termN*: This emulates an evaluation of a multi-term Boolean AND query on the index with document-at-a-time query. The index ordered by increasing document IDs should be used in this query. Note again, Java’s build-in intersection methods are NOT allowed in this assignment. Output format is the same.

  For example:

  ```
  FUNCTION: docAtATimeQueryAnd query_term1, …, query_termN
  xx documents are found
  yy comparisions are made
  zz seconds are used
  Result: 100, 200, 300… (list the document IDs)
  ```

  Should display “terms not found” if it is not in the index.

- **docAtATimeQueryOr** *query_term1, ..., query_termN*: This emulates an evaluation of a multi-term Boolean OR query on the index with document-at-a-time query. The index ordered by increasing document IDs should be used in this query. Output format is the same.

  For example:

  ```
  FUNCTION: docAtATimeQueryOr query_term1, …, query_termN
  xx documents are found
  ```

```
          yy comparisions are made
          zz seconds are used
          Result: 100, 200, 300… (list the document IDs)
```

Should display “terms not found” if it is not in the index.

Your main function should be named “CSE535Assignment.java”. It should take the index file as the first parameter when starting. It should be able to execute the functions mentioned above and write corresponding results into a log file with the required format. The log file name should be the second parameter to your main function. The third parameter is an integer that will be used in the **getTopK**. The last parameter is a file contains query terms. In this file, each line contains a set of query terms that are separated by blank spaces. An example query term file is given in the following (a sample query term file is also provided along with this description):

```
#################################################################################
     query_term1 query_term2
     query_term3 query_term4 query_term5
#################################################################################
```

Each set of query terms will trigger an execution of the **getPostings**, **termAtATimeQueryAnd, termAtATimeQueryOr**, **docAtATimeQueryAnd** and **docAtATimeQueryOr** once. For an example, if we have the aforementioned two-line example query file, you should record the outputs in the following order.

getTopK *K* (Note here, the getTopK function only need to run once)

getPostings query_term1

getPostings query_term2

termAtATimeQueryAnd query_term1, query_term2

termAtATimeQueryOr query_term1, query_term2

docAtATimeQueryAnd query_term1, query_term2

docAtATimeQueryOr query_term1, query_term2

getPostings query_term3

getPostings query_term4

getPostings query_term5

termAtATimeQueryAnd query_term3, query_term4, query_term5

termAtATimeQueryOr query_term3, query_term4, query_term5

docAtATimeQueryAnd query_term3, query_term4, query_term5

docAtATimeQueryOr query_term3, query_term4, query_term5

Function **getTopK** needs to be executed only once, no matter how many sets of query terms we have. In summary, your program will start running by executing the following example command:

```
java CSE535Assignment term.idx output.log 10 query_file.txt
```

IMPORTANT NOTE: you do NOT have to implement Java methods that named exactly as getTopK(), getPostings(), and etc. As long as your code fulfills the required functionality and generates the correct output with the correct format, you are fine.

Log File Format

During the grading, your code will be compiled, executed and the log file will be generated. Then the log file will be analyzed and graded accordingly.

Your log file should follow the following format (wrong format will result in 0 in the auto-grading). A sample output file is also provided along with this description:

```
###########################################################################################

FUNCTION: getTopK 10 (Note here again, the getTopK function only need to run once)
Result: term1, term2, term3, …, term10 (list the terms)
FUNCTION: getPostings query_term1
Ordered by doc IDs: 100, 200, 300… (list the document IDs ordered by increasing document IDs)
Ordered by TF: 300, 100, 200… (list the document IDs ordered by decreasing term frequencies)
FUNCTION: getPostings query_term2
Ordered by doc IDs: 100, 200, 300… (list the document IDs ordered by increasing document IDs)
Ordered by TF: 300, 100, 200… (list the document IDs ordered by decreasing term frequencies)
FUNCTION: termAtATimeQueryAnd query_term1, query_term2
xx documents are found
yy comparisions are made
zz seconds are used
Result: 100, 200, 300…
FUNCTION: termAtATimeQueryOr query_term1, query_term2
xx documents are found
yy comparisions are made
zz seconds are used
Result: 100, 200, 300…
FUNCTION: docAtATimeQueryAnd query_term1, query_term2
xx documents are found
yy comparisions are made
zz seconds are used
Result: 100, 200, 300…
FUNCTION: docAtATimeQueryOr query_term1, query_term2
xx documents are found
yy comparisions are made
zz seconds are used
Result: 100, 200, 300…
(Similar for the second set of queries: query_term3, query_term4 and query_term5. Details can be found in the
sample output file.)
###########################################################################################
```

Evaluation

A successful implementation should be able to support all the aforementioned functions, generate the correct results for the query terms, and write the output to the log file in the required format.

About the running time: your program should be able to finish the loading, querying and flushing output within minutes. It will not be evaluated if times out.

Points will be awarded for proper functioning of the program, and each of the methods you are asked to implement. More points are awarded for proper functioning of the two scoring methods.

What to Submit

You should submit all source codes (i.e., ".java" files, not the index data or any intermediate data) using cse-submit script.