CSE 435/535 Information Retrieval (Fall 2018)
Project Two: Boolean Query and Inverted Index

Due Date: October 18th 2018, 23:59 pm (EST)

Overview

In this project, you will be given Lucene index generated from the PubMed corpus [1]. Based on this provided index, you task is to build your own inverted index using the information extracted from the given data. Your index should be stored as Linked List in memory as the examples shown in textbook (Refer Chapter 1 – Boolean Retrieval). Having built this index, now you are required to implement two strategies to return boolean query results: Term-at-a-time (TAAT) and Document-at-a-time (DAAT). Your implementation should be based on **Java**.

Input Dataset

The given lucene index is generated by indexing a subset of PubMed corpus on Solr (which is similar to what you have done in project part one). Basically, it contains article titles in three languages: French, Spanish, and English. Corresponding field names are: text_fr, text_es, and text_en.

Each document contains only ONE of the above text fields (text_fr/text_es/text_en) and is assigned with a document identifier (id). Some example documents:


```
    "id": 3,
    "text_es": "Eugenesia y Falange a traves de la revista Ser (-)."
}, {
    "id": 4,
    "text_es": "Novedades, críticas y propuestas al DSM-: el caso de las disfunciones sexuales, la disforia de género y
}, {
    "id": 5,
    "text_es": "Trastorno depresivo, trastorno de ansiedad y dolor crónico: múltiples manifestaciones de un núcleo fisi
```


Please find the lucene index for this project from here: https://piazza.com/buffalo/fall2018/cse435535/resources (Under Projects)

Step 1: Interact with Lucene Index and Build Your Own Inverted Index

As the examples in referred textbook, postings lists should be stored as **Linked Lists**.

You will need to construct your own index in which postings of each term should be ordered by **increasing document IDs**. For example:

term1	doc1	doc2	doc4
term2	doc2	doc4	doc7

In order to use the APIs to talk to the given index and get the information you need, import the Lucene module to your java project. The module ("lucene-core-7.4.0.jar") can be downloaded from here: https://lucene.apache.org/core/ .

Note:

- You are supposed to ONLY use the methods included in package **org.apache.lucene.index** to get information for your index. DO NOT use other classes which can answer the queries directly, such as IndexSearcher.

- The documentation for IndexReader can be found here: http://lucene.apache.org/core/7_3_0/core/org/apache/lucene/index/IndexReader.html

Step 2: Boolean Query Processing

You are required to implement the following methods, and provide the results for a set of boolean queries (AND/OR) **on your own index**. Results should be output as a .txt file in the required format.

1. Get postings lists

This method retrieves the postings lists for each of the given query terms. Input of

this method will be a set of terms: term0, term1,..., termN. It should output the postings for each term in the following format:

GetPostings
term0
Postings list: 1000 2000 3000 … (by increasing document IDs)
GetPostings
term1
Postings list: 1000 3000 6000 … (by increasing document IDs)
…
GetPostings
termN
Postings list: 2000 7000 8000 … (by increasing document IDs)

2. Term-at-a-time AND query

This method is to implement multi-term boolean AND query on the index using term-at-a-time (TAAT) strategy. Input of this function will be a set of terms: term0, term1, …, termN. Output format of this method should be:

TaatAnd
term0 term1 … termN
Results: 1000 2000 3000 … (by increasing document IDs)
Number of documents in results: x
Number of comparisons: y

If the result of this query is empty then output:
TaatAnd
term0 term1 … termN
Results: empty
Number of documents in results: 0
Number of comparisons: y

3. Term-at-a-time OR query

This function is to implement multi-term boolean OR query on the index using TAAT. Input of this function will be a set of query terms: term0, term1, … , termN. Output format of this method should be:

TaatOr
term0 term1 … termN
Results: 1000 2000 3000 … (by increasing document IDs)
Number of documents in results: x
Number of comparisons: y

NOTE: A comparison (for field "Number of comparisons") is counted whenever you compare two Document IDs during the union or intersection operation.

4. Document-at-a-time AND query

This function is to implement multi-term boolean AND query on the index using document-at-a-time (DAAT) strategy. Input of the function will be a set of query terms: term0, term1, …, termN. Output the following to the output file:

DaatAnd
term0 term1 … termN
Results: 1000 2000 3000 … (by increasing document IDs)
Number of documents in results: x
Number of comparisons: y

If the result of the query is empty then output:

DaatAnd
term0 term1 … termN
Results: empty
Number of documents in results: 0

Number of comparisons: y

5. Document-at-a-time OR query

This function is to implement multi-term boolean OR query on the index using DAAT. Input of this function will be a set of query terms: term0, term1, …, termN. Output the following to the output file:

DaatOr
term0 term1 … termN
Results: 1000 2000 3000 … (by increasing document IDs)
Number of documents in results: x
Number of comparisons: y

IMPORTANT: For AND operations (both Term-at-a-time AND query and Document-at-a-time AND query), you are required to implement the list intersection via *evenly spaced Skip pointers*. In other words, if the positing list is of size *P*, then use √P evenly spaced skip pointers. For further details, please refer subsection (Faster postings list intersection via skip-pointers) of Chapter 2 from the text book.

There is no requirement for the names of your Java methods. However, you should submit a .jar file **named exactly as "UBITName_project2.jar"**, and your program should start running by executing the following command **on Timberlake**:

java -jar UBITName_project2.jar path_of_index output.txt input.txt

Here, the first input parameter *path_of_index* should be able to accept a string indicating the path of the given Lucene index. The second input parameter *output.txt* refers to the output file, while the third one, *input.txt* refers to the input file which contains the query terms.

The following assumptions can be made:

- The number of input query terms can be varied.

- All of the input query terms are selected from the vocabulary.
- Query terms should be processed in the order in which they are written in the query. Say, you should process term0 first, then term1, so on and so forth.
- Input query terms will be separated by 'one blank space' in the input file. You can assume there is no blank space within any query term.
- DO NOT use Java build-in methods to do unions and intersections on postings lists directly. DO NOT use build-in functions to check the presence of document IDs directly, such as List.contains(), when you are performing union/intersection on postings. DO NOT use classes such as IndexSearcher to answer queries directly. Create your own pointers and have fun!
- Output should be formatted exactly the same as required. Otherwise you will not be able to get credits because grading will be automated!

Sample Input Output

The input file will be a .txt file containing multiple lines. Each line refers to a set of query terms which are separated by one blank space. For example:

term0 term1
term2 term3 term4 term5 term6 term7
term8 term9 term10

In this case, the output should be in the following order:

GetPostings term0 term1
TaatAnd term0 term1
TaatOr term0 term1
DaatAnd term0 term1
DaatOr term0 term1
GetPostings term2 term3 term4 term5 term6 term7
TaatAnd term2 term3 term4 term5 term6 term7
TaatOr term2 term3 term4 term5 term6 term7

DaatAnd term2 term3 term4 term5 term6 term7
DaatOr term2 term3 term4 term5 term6 term7
GetPostings term8 term9 term10
TaatAnd term8 term9 term10
TaatOr term8 term9 term10
DaatAnd term8 term9 term10
DaatOr term8 term9 term10

The corresponding output file for the input mentioned before will be:

GetPostings
term0
Postings list: 1000 2000 3000 …
GetPostings
term1
Postings list: 1000 2000 3000 …
TaatAnd
term0 term1
Results: 1000 2000 3000 …
Number of documents in results: x
Number of comparisons: y
TaatOr
term0 term1
Results: 1000 2000 3000 …
Number of documents in results: x
Number of comparisons: y
DaatAnd
term0 term1
Results: 1000 2000 3000 …
Number of documents in results: x
Number of comparisons: y
DaatOr
term0 term1

Results: 1000 2000 3000 …
Number of documents in results: x
Number of comparisons: y
GetPostings
term2
Postings list: 1000 2000 3000 …
GetPostings
term3
Postings list: 1000 2000 3000 …
GetPostings
term4
Postings list: 1000 2000 3000 …
GetPostings
term5
Postings list: 1000 2000 3000 …
GetPostings
term6
Postings list: 1000 2000 3000 …
GetPostings
term7
Postings list: 1000 2000 3000 …
TaatAnd
term2 term3 term4 term5 term6 term7
Results: 1000 2000 3000 …
Number of documents in results: x
Number of comparisons: y
TaatOr
term2 term3 term4 term5 term6 term7
Results: 1000 2000 3000 …
Number of documents in results: x
Number of comparisons: y
DaatAnd
term2 term3 term4 term5 term6 term7

Results: 1000 2000 3000 …
Number of documents in results: x
Number of comparisons: y
DaatOr
term2 term3 term4 term5 term6 term7
Results: 1000 2000 3000 …
Number of documents in results: x
Number of comparisons: y
GetPostings
term8
Postings list: 1000 2000 3000 …
GetPostings
term9
Postings list: 1000 2000 3000 …
GetPostings
term10
Postings list: 1000 2000 3000 …
TaatAnd
term8 term9 term10
Results: 1000 2000 3000 …
Number of documents in results: x
Number of comparisons: y
TaatOr
term8 term9 term10
Results: 1000 2000 3000 …
Number of documents in results: x
Number of comparisons: y
DaatAnd
term8 term9 term10
Results: 1000 2000 3000 …
Number of documents in results: x
Number of comparisons: y
DaatOr

term8 term9 term10
Results: 1000 2000 3000 …
Number of documents in results: x
Number of comparisons: y

Grading and Evaluation

A successful implementation should be able to support all the functions mentioned before. It should also generate correct results in the required format. Failure in following the instructions and requirements will result in 0.

Rubrics:
Total points for this project: 15
Successfully rebuild index and correct output for GetPostings: 5
Correct strategy for TAAT: 3
Correct strategy for DAAT: 3
Correct results for TaatAnd / TaatOr / DaatAnd / DaatOr: 1 + 1 + 1 + 1

What to Submit

1) You should use cse-submit script to submit a .jar file named exactly as "**UBITName_project2.jar**". Double check and make sure your submission can be executed successfully **on Timberlake**. You need to use either "submit_cse435" (undergrad) or "submit_cse535" (grad), depending on your registration status. If you haven't used it, the instructions on how to use it is here: https://wiki.cse.buffalo.edu/services/content/submit-script

2) You should submit a zipped (.zip ONLY) version of your source code named exactly as "**UBITName_project2.zip".** Please do not use any other compression tool other than zip, i.e., no 7-Zip, no RAR, etc.

NOTE: Late submissions will NOT be accepted. The deadline is firm (i.e., October 18th 2018, 23:59 PM (EST)), if your timestamp is 12:00 AM, it is a late submission. Please start early.

References

[1] https://www.ncbi.nlm.nih.gov/pubmed/

FAQ's

1. **How should I get started for this project?**
   - First, make yourself familiar with fundamental concepts such as dictionary, postings, Inverted Index and Boolean operations. The best place to start is reading thoroughly the lecture slides and Chapter 1,2 of the referred textbook.

2. **How much programming expertise is needed to complete this project?**
   - You are expected to know the fundamentals of object oriented programming and be familiar with basic data structures such as Queue, Heap, HashMaps, Lists and so on.

3. **What is PubMed?**
   - PubMed is a bibliographic repository that contains articles primarily from life sciences and biomedicine. At present, it contains more than 5,200 worldwide journals in about 40 languages.

4. **How should I handle the special characters present in French language?**
   - You should learn about the different kinds of character encoding (a good place to start is APPENDIX of project 1) and research on how to handle them in java.

5. **My program takes a while to execute. Would that be a problem in grading?**
   - Although the focus of this project is to test the correctness, we encourage you to be mindful of the data structure you are using for implementation. Unless it takes an unreasonably long time, you are fine in terms of grading but again please carefully analyze your code.