Information Retrieval CS

ASSIGNMENT 2

In this assignment you will create inverted index on provided corpus.
You will work in group or 2 (and only 2) members.
Assigned on 13th March, 2019 Due date: 3rd April, 2019

Corpus:

The corpus contains 3495 documents of type HTML. The documents are divided in three sub-directories (1-3).
You cannot change the names and directories of these documents.
Link of the corpus: \\cactus\xeon\Spring 2019\Noshaba Nasir\Information Retrieval

TASK 1: Preprocessing:

The first step in creating an index is tokenization. You must convert a document into a stream of Tokens suitable for indexing. Your tokenizer should follow these steps:

1. Accept a directory name as a input, and process all files found in that directory
2. Extract the document text with an HTML parsing library, ignoring the headers at the beginning of the file and all HTML tags
3. Split the text into tokens (You can use libraries available in python/java to do so)
4. Convert all tokens to lowercase (this is not always ideal, but indexing intelligently in a case-sensitive manner is tricky)
5. Apply stop-wording to the document by ignoring any tokens found in this list (stop words given in stoplist.txt on cactus)
6. Apply stemming to the document using any standard algorithm – Porter, Snowball or KStem stemmers are appropriate. You should use a stemming library for this step.

The resulting tokens will be used in next step

TASK 2: Creating Inverted Index:

Step 1:

Receive the token in <term, docID> pair from step2 and create the inverted index in memory using hashmaps in java or dict in python.

The corpus is divided in three subdirectories. You should treat each sub-directory as a block, you will create an index on one block at a time (you will be penalized if you create index of all document in one go).

Step 2:

After creating index of one block (sub-directory) write it in file using following format for each term

<term t>,<df_t>,<docID>,<tf_{td}>,<pos1>,<pos2>,<pos3>…,<docID>,<tf_{td}>,<pos1>,<pos2>,<pos3>…,

For example, considering the 4 document in fig 5.1 in CMS the posting list for term *fish* will be
fish,4,1,2,2,4,2,3,7,18,23,3,2,2,6,4,2,3,13

The posting list should be sorted on base of docID and further on positon.

At the end of this step you should have 3 index files named as index_1.txt, index_2.txt, index_3.txt, where 1, 2, 3 represent the name of subdirectory on which index was created.

TASK 3: Merging Inverted Indexes:

In this part you will write a module that will take file path of two indexes and will merger them in one index. The merged index will be written on file in same format as individual index.
You will use buffer readers and writers to perform this task, complete index file should not be read/write at once.

Use this module to merge the three indexes create in previous step. Final index should be on a file named
`inverted_index.txt`

TASK 4: Keeping additional Information:

As required in previous step, each document should have a unique integer docID be. docID should be assigned to the document at the time of preprocessing. You will also need to store document length which will be used in next assignment (Ranked retrieval).

Keep a separate file that maps sub_directory/documentName to docID and its document length, named docInfo.txt. Format of this file should be as follow.

`docID,sub_directory/documentName,documentLength.`

TASK 5: Encoding posting Lists:

In this task, you will build a compressed index using gap encoding with variable byte encoding for each gap. The dictionary and postings file will also be separate in this task.

Step 1:

After creating index in memory like in step 1 of task 2, encode the posting list using gap encoding with variable byte encoding (for more details See Section 5.4.4 and 5.4.5 in CMS).

Step 2:

Write <term>, <posting file offset> in index_vocab.txt file

Write the posting lists of each term in index_postings.txt file

Where <posting file offset> is the position in index_postings.txt file where the posting list of <term> starts.

NOTE:

The index built in this task will also create one index per sub_directory. Your merging index module should also work on this index.

TASK 6: Reading Inverted Indexes:

In this task you will perform Boolean retrieval. You will get a query as input, tokenize the query in same way as in task 1.

Your program should print to the list of documents containing the query terms, one document file name on each line in ascending lexicographical order in following format

Information Retrieval CS
`Sub_directory/documentName`

If no result is found print

```
No result found
```

This module should also work on both indexes (compressed and uncompressed).

Results and Report:

Write a brief report of following aspects of your assignment:

1. A brief summary of structure of your program. Also specify which libraries you used in each step if any.
2. Report the time consumed in each task.
3. Time consumed in task 6 if you were retrieving data from compressed is uncompressed index.
4. Compare the performance of retrieval from compressed index if vocab and postings were both in memory vs when only vocab is in memory and postings are on disk.
5. Write a conclusion by giving your thoughts on what could be done differently to improve the indexing process.

Submissions:

Submissions will be on Slate, you will only submit your code and report (no index files). One one group member should make a submission.