Comp 479/6791 Project 2 — updated version Fall 2020

Objectives: Implement the naive indexer. Implement single term query processing. Implement and compare lossy dictionary compression.

Due date: October 21, 2020

Data: Use Reuters21578. For docID, use the NEWID values from the Reuters corpus to make your retrieval comparable. Make sure you access the sgm files and the docIDs in **ascending order**, this requires an extra step in Linux.

Description:

Subproject I: naive indexer

1. develop a module that while there are still more documents to be processed, accepts a document as a list of tokens and outputs term-documentID pairs to a list F. Punctuation is not considered to be a token by itself, but your tokenizer might recognize tokens that include punctuation signs
2. when there is no more input, sort F and remove duplicates
3. turn the sorted file F into an index by turning the docIDs paired with the same term into a postings list
4. find a way to determine, how much time your index takes to compile (not graded in this assignment)

Subproject II: single term query processing Implement a query processor for single term queries, i.e. queries that consist exclusively of one term

1. Input: one term
2. Output: append term to postings list. Make sure that the postings list is ordered
3. turn your output into json format:

   Listing 1: Python example

   ```
   import json
   #Submit your queries in the following format:
   # 1. <query1> -> replace with the corresponding query string
   # 2. [1,2,4] -> replace with the corresponding postings list
   # 3. change the filename "sampleQueries.json" for the appropriate situation
   q = {"<query1>": [1,2,4],"<query2>": [2,3],"<query3>": [1,4]}
   json.dump(q, open("sampleQueries.json","w",encoding="utf-8"),indent=3)
   ```

4. validate query returns for three sample queries (you have to decide on your sample queries)

Subproject III: implement lossy dictionary compression, 'recreate' Table 5.1 columns 1 and 2

1. implement the lossy dictionary compression techniques of Table 5.1 in the textbook and compile a similar table for Reuters-21578 for dictionary and non-positional index. Are the changes similar? Discuss your findings. (Note that stemming is not required here, if you run out of time before you get the Porter stemmer to work, that is ok for this assignment, the remaining table is fine.) To use a unified list of 30 and 150 stop words, use `https://teacherjoe.us/Vocab200.html` (first 30 and first 150 words respectively)
2. compare retrieval results for your three sample queries of Subproject II when you run them on your compressed index. Discuss your findings in your report

Deliverables:

1. individual project
2. well documented code
3. "sampleQueries.json": output of your three sample queries in json format as described
4. "challengeQueries.json": output of the queries posted two days before the deadline in json format as described. Run queries on both indices
5. any additional testing or aborted design ideas that show off particular aspects of your project
6. a project report that summarizes your approach, illustrates your designs, presents your table of savings for lossy dictionary compression and discusses, what you have learned from the project

https://teacherjoe.us/Vocab200.html

Marks:

Naive indexer implementation	2pts	Attr5
Resulting inverted index	1pt	Attr4
Single keyword sample query results	1pt	Attr5
Challenge single keyword query results	1pt	Attr4
Dictionary compression implementation	1pts	Attr5
Dictionary compression table	1pts	Attr5
Report	1pt	Attr6