COL-764 Assignment 1
Inverted Index Construction
Submitted by-
Dipanshu Patidar
2018CS50405

Introduction:

This report documents the implementation details related to the Corpus Pre-processing, Inverted Index Construction, Compression Strategies, Mode of Query Retrieval etc. used in the assignment which when integrated constitutes a full Boolean Retrieval System.
With the help of sub-headers, I would describe the implementation details regarding all of these processes and would also mention how the various functions used in the code achieve their respective tasks.

Composition of Code files:

invidx_cons.py - This is the file related to the construction of the Inverted Index, by first processing the documents and then storing them on the disk in a compressed way.

query.py – This file preforms the query processing and retrieval tasks.

Operations done in invidx_cons.py file

1)Pre-Processing(PROCESS FINCTION IN INVIDX_COONS.PY) :
1. Parsing the xml content was done:
Using beautifulsoup4, all the DOC tags in a file in a file are parsed. These individual documents are thenfurther processed by bs4 to get the text present in all the required tags l isted in xml_tags_info.
2. Tokenization:
Using re.split() function which can tokenize a string based on multiple delimiters. We store the all the tokens of a particular document (each having a unique DOCNO) in a Set of words. Set helps us to avoid same docno being addded to the posting list of the same word more than once.

3. Processing the set of tokens to remove Stopwords and stem them using PorterStemmer:

The set of words obtained in the previous step is now processed to remove the stopwords (listed in the stop wordfile) and stem them using the PorterStemmer to get the root form of individual words in that particular set

After preforming these pre-processing tasks for a single file in the collection (which may contain multiple DOCs) , we obtain the set of termsin their root form with their list of docids they are present in.

2)Further processing:

4. store the data of each word and their posting list in the disk using many chunk files.
5. read the disks and compress posting lists for each word according to the given compression type. We are storing the following information in indexfile.dict file:
    a. compression type
    b. next lines are of the following format: word starting_index_in_idx_file length(in bytes) until ‘&’ appears in the whole line
    c. next lines are the docnos where ith docno has a mapping to no. `i`
    d. docno’s continue until `&` appears.
    e. now remaining last line contains space separated stopwords.
6. Then we are compressing the data according to the compression strategy.
7. Gap encoding is done before compression.

So basically, we are storing the following information in indexfilename.dict file:

1. compression type
2. starting byte and length for posting lists of each word.
3. mapping of docno to integer indices.
4. stopwords

3)Operations in query.py:

1. Reading indexfilename.dict file and getting compression type using the first read line.

2. Store the dictionary words and their starting byte in idx file with their length in the memory.
3. extract mapping of docno to index integers.
4. extract stopwords
5. read queries
6. for each single word query: * check if the word is in stopwords, if yes, return null * if word is not in stopwords, decode its posting list with the respective compression type and return the result.
7. for each multiple word query: * check if the word is in stopwords, if yes, move to the next word * if word is not in stopwords, decode its posting list with the respective compression type and if it is the first word of the query, store its posting list to final list and move to the next, else take intersection from the final posting list.
8. print the results

Metrics of Interest:

The following metrics are being evaluated on the dataset provided and the sample query file (consisting of 50 queries) given.

1. ISR :

- C0- 0.370
- C1- 0.112
- C2- 0.091
- C3- 0.141

2. Compression Speed:

- C0- 844754.32 milli-sec
- C1- 645840.77 milli-sec
- C2- 674746.63 milli-sec
- C3- 613669.64 milli-sec

3. Query Speed:

- C0- 0.08081* 10^6 micro-sec/query
- C1- 0.0512* 10^6 micro-sec/query
- C2- 1.5123 * 10^6 micro-sec/query
- C3- 0.0503* 10^6 micro-sec/query