Assignment #2

Inverted Index : Inverted index is a mapping from content(words in this case) to their location in document. The purpose of this is to provide rapid full word searches at the cost of higher processing time at the time word is entered in the inverted index.

Process:

1. **Decompress:** To start with we are given a sum total for more than 3 million documents. The documents have been compressed using gzip so first we decompress this using "java.util.zip.GZIPInputStream" library. This allows us to uncompress the file in tsv(tab separated value) format. Note: I was unable to download the TREC format suggested as the download keeps on failing halfway through. Gzip is a lossless compression, so we can reconstruct original data.

2. **Program logic:**

- In this code we will do a double pass of data. The first pass is to estimate the barrel size for lexicon partitioning while generating postings. In this pass we create a HashMap (frequencyGraph) and calculate frequency of each word as well as the number of documents in which the word occurs. This allows us to more precisely maintain allot a dynamic barrel size for each lexicon. We will be maintaining this table in the main memory. For all the terms this table seems to occupy around 1 GB of main memory.

- Next is Term to TermID mapping: We are maintaining a HashMap(termIDS) which maps terms to term IDs. We are allotting term id sequentially. Each term has a unique term id. We are maintaining this data structure in main memory. Using Integer instead of string in the lexicon table allows for compression.

- Next is MongoDB database which maintains document text mapped to document ID, this allows us to query data at last and highlight relevant(search query) data. We are doing this by forming a simple database URL and a collection in it called URLCollect. We are filling the database during the first pass of data.

- Next is Lexicon data which is stored in a HashTable. The key for the hash table is the termID, which we get from HasMap TermIDS. While the value is an array of "Document frequency" i.e. number of documents the word has appeared in at least

once and the location in the inverted index file where the postings for that occurrence is stored.

3. **The inverted Index Table:**

- We can use RandomAccessFile to write to and read from any point the inverted index file.
- We also fills the lexicon HashMap in which barrel size is determined by the frequency details we gathered during the first pass.
- We iterate through each word in the collection of documents. We then split document ID, URL, title and data on the basis of tab(\t).
- We check if the word is in termsIDS HashMap. If not then, we assign it a new term ID sequentially. We add all the words to two HashMap,
    1. The wordposition has the term as key and an integer array which contains the position of the occurrences of that term in the document.
    2. The mapforpage HashMap which contains frequency of occurrence of each word.
- After iterating a document we have a Map containing each word that appears in it and another Map containing locations of terms each occurrence.

  We store data in the form

DocID	frequency	position1	position2	position3	position4

  Difference postings are delimited by **" > " (greater than),** while (DocID,frequency) pairs and positions are delimited by a **" :" (a colon).** DocID and frequency is delimited by a comma and so are different positions.
  Which would look like:
  docID,frequency:locations1,location2,location3>docID2,frequency2:locations1,location2,location3;x

  x denotes end for expression

- We start by iterating mapforpage Hashmap, for each term we find the relevant termID in the termIDS map. From this termID we find the location data in the

lexicon HashTable. We then seek this address using RandomAccessFile and read a number of bytes based on frequencyGraph. We process the bytes to string. If the string is empty we insert the DocID,frequency pair, getting frequency from mapforpage. This is followed by inserting position data by searching the key in wordposition Map. we add all the locations and delimit it with a **colon**.
If the String is not empty we split the data according to aforementioned delimiters and put it into a temporary HashMap foreachDocument with term as key and an integer array. The first element of the array is the frequency and the remaining the position. We also insert data from mapforpage and wordposition in the temporary hashmap. Now we sort the temporary HashMap according to the frequency in descending order(first element of value). This allows us to store documents with most frequent occurrence at the forefront. Now we insert back these elements.

- The HashTable *lexicon* and HashMap *termIDS* and *frequencyGraph* are stored in file “lexicon”,”termID” and “frequency” respectively.

5. **For querying data:**

- We read HashTable *lexicon* and HashMap *termIDS* and *frequencyGraph* from file “lexicon”,”termID” and “frequency” respectively.
- For querying data we are provided with a search query, we lookup this query term in the termIDS map and find corresponding term ID. We then look at the termID in the lexicon map and find the address of posting information in the inverted index file. We read the data (how much according to frequencyGraph), split it according to the delimiters set and get the first 10 results. Since these postings were sorted during time of insertion we don't have to sort it. Just take the top 10 results, query DocID in the MongoDB database to find the data stored and use location data to highlight the query where we found it.

Search term “there” being encolde by ←** **---->