Contents

To run the project, open the project **wse-hw** in IntelliJ.

1. Index Structure and Generation Process

Structure

Inverted Index

Inverted list: docIDs freqs docIDs freqs docIDs freqs…
Inverted index is formed by consecutive inverted lists.
Metadata is stored in another file.

Metadata

Compressed and stored in a separate file.

```
int[] lastDocIds;
```

```
long[] docIdBlockStarts;
int[] docIdBlockSizes;
long[] freqBlockStarts;
int[] freqBlockSizes;
```

Lexicon

Key: String term
Value: A class called Lexicon:

```
long startOffset;// Starting position of the inverted list
long endOffset;// Ending position of the inverted list
int numBlocks;// Number of blocks
int docFrequency;// Total number of doc containing the term
long metadataOffset;// Starting position of the metadata of the inverted list
```

PageTable

A list of DocumentInfo:

```
int docID;
String url;
int termCount;
```

What the Program Can Do

1. Parsing the source file into sorted temp files, each line in the temp files is a posting "term docId frequency".
   This will generate several temp files and a file **document_info.ser**, which is the serialized pageTable.
2. Merging the postings in temp files into one large file in which each line is **Term docID Frequency**.
3. Reformatting and compressing the large file into inverted index, generating metadata into a separate file. This will generate the inverted index file **inverted_index.bin**, the metadata file **metadata.bin** and the lexicon **lexicon.bin**.

How does the Program Run Internally

Parsing

1. Iterate the source file, read one line at a time.
2. When meeting **TEXT** tag, we are inside the text.
3. The first line inside text is the **URL** of the document.
4. Inside the text of each document, we extract the terms and maintain their frequencies inside the document in a HashMap.
5. When meeting **\TEXT** tag, we are at the end of the text. Convert the HashMap into postings with format **""Term docID Frequency""** and

put them into a **buffer**. If the buffer is large enough, write it to a temp file. Also save the document info **""docID, URL and Total number of terms in the document""** into **PageTable**

Merging

1. Create a BufferedReader for each temp file.
2. Create a BufferedWriter to write into the merged large file.
3. Use IO Efficient Merge Sort.

Reformatting

1. Use block size of 128.
2. Compress the docID and freq blocks and write into file when block is full or meeting a new term.
3. Compress and write metadata when meeting a new term.
4. Maintain the starting and ending offset, number of blocks, total number of documents containing of the previous term and put into Lexicon when meeting a new term.

How to Run the Program

Open the project **wse-hw** in IntelliJ.

Parsing

- Run `parsing(String srcFilePath, String tempFilePath, String dataFilePath)`
- It's in **src/component/component1/Parse.java** in module **hw2**.

Merging

- Run `mergeSortedTempFiles(String tempFilePath, String dstFilePath)`
- It's in **src/component/component2/Merge.java** in module **hw2**.

Reformatting

- Run `convertToCompressedIndex(String sortedFilePath, String invertedIndexPath, String metadataPath, String lexiconFilePath)`
- It's in **src/component/component2/Merge.java** in module **hw2**.

How Long It Takes on the Provided Data Set

- Parsing 18min
- Merging 42min
- Reformatting to resulting index file 1.5hr

How Large the Resulting Index Files Are

Index file 2.62 GB on disk.
Metadata file 333.1 MB on disk.

Limitations

I didn't store the start and end offset of each document in the source file which influences the snippet generation.

2. Query Processor

What the Program Can Do

Perform AND and OR semantic query processing.

How does the Program Run Internally

Query Processor

1. Load Lexicon, PageTable.
2. Create readers for metadata and index files.
3. Call corresponding query processor based on query type AND/OR.

AND Semantic

1. For each term in the query, **openList()** decompress the metadata of each term. And create an instance InvertedList for each term.

```
public class InvertedList
    String term;
    Metadata metadata;
    int targetBlockIdx;
    int targetIdxInBlock;
```

These two are for **getFreq()**:

```
int targetBlockIdx;
int targetIdxInBlock;
```

2. Sort the inverted lists based on the number of postings.
3. In a while loop, **did** is the docID of the document in which all terms occur, did starts from 0 to the max docID of the shortest list.
4. Get the next docID greater or equal to did from the shortest list and update did using **did = nextGEQ(did)** .
5. Iterate the other lists, maintain a variable **d= nextGEQ(did)** for each list to see if **d = did**. If **d > did**, update **did = d**, and go back to Step 4. If **d = -1**, it means there's no more intersection, we can break the loop. If **d = did**, move to the next list.

6. After iterating all the other lists, if **d = did**, it means we have found a document containing all the terms in the query.
7. Compute the impact score of the document.
8. Add the document to the heap if the size of heap is less than the results size or the heap's peek's score is less than the document's score. Go back to Step 4.
9. Return the heap.

OR Semantic

1. For each term in the query, **openList()** decompress the metadata of each term. And create an instance InvertedList for each term.
2. Create a hash table to record the impact score of every document.
3. For each list, insert or update its entry in the hash table.
4. Return the top-k results by iterating the hash table and insert into a min heap if the heap size is less than k or the impact score of current document is higher than the top document in the heap.
5. Return the heap.

How to Run the Program

1. Open the project **wse-hw** in IntelliJ.
2. Go to **src/Test.java** in module **hw3**.
3. Run the main function.

Limitations

After decompress a block, I didn't store it in memory. So I have to decompress it every time when I'm iterating the docIDs inside the block.