**HW 2**

By Ziming Song

[zs2815@nyu.edu](mailto:zs2815@nyu.edu)

# **what program can do**

This program can setup, sort and build a Search Engine Index based on compressed *.trec.gz* data set.

This program will generate:

## **1. Inverted Index Structure**

*word0: {docID0, freq0}, {docID1, freq1}, …, {docIDn-1, freqn-1}*

*word1: {docID0, freq0}, {docID1, freq1}, …, {docIDn’-1, freqn’-1}*

*…*

*wordn-1: {docID0, freq0}, {docID1, freq1}, …, {docIDn\*-1, freqn\*-1}*

## **2. Lexicon Structure**

*word0: {begin\_pointer0, end\_pointer0, docID\_nums0}*

*word1: {begin\_pointer1, end\_pointer1, docID\_nums1}*

*…*

*wordn-1: {begin\_pointern-1, end\_pointern-1, docID\_numsn-1}*

## **3. Page (docID-to-URL) Table**

*{docID0, docNO0, dataLen0, word\_num0, URL0}*

*{docID1, docNO1, dataLe10, word\_num1, URL1}*

*…*

*{docIDn-1, docNOn-1, dataLenn-1, word\_numn-1, URLn-1}*

All the structures and table mentioned above can be written in bit or text format file according to your configuration. The result of index file is compressed.

# **how to run the program**

The program is written in C++. You have to install and configure cmake first.

1. change “config.h” to configuration filepath, folder path. The meaning of config parameters are listed below:

|  |  |
| --- | --- |
| parameter | meaning |
| DATA\_SOURCE\_PATH | File path of “\*.trec.gz” |
| INDEX\_FILE\_FOLDER\_PATH | Folder path you want to keep temporary Index files |
| FINAL\_INDEX\_PATH | File path of final merged Inverted Index file |
| LEXICON\_PATH | File path of Lexicon file |
| DOC\_TABLE\_PATH | File path of document table file(or page table file) |
| FILEMODE | Write file in bit format or text format. Value should be set to 1(bit) or 0(text). |
| IS\_DEBUG | Indicate whether output debug information in terminal. Value should be set to 1 or 0. |
| IS\_INDEX | Indicate whether reindex input trec data. Value should be set to 1 or 0. If value is 0, it will merge files in INDEX\_FILE\_FOLDER\_PATH |
| IS\_MERGE | Indicate whether merge files in INDEX\_FILE\_FOLDER\_PATH. Value should be set to 1 or 0. |
| IS\_WRITE\_PAGE | Indicate whether write page table into file. Value should be set to 1 or 0. |
| IS\_WRITE\_LEXICON | Indicate whether Lexicon Structure into file. Value should be set to 1 or 0. |
| IS\_DELETE\_TEMP | Indicate whether delete temporary Index Inverted file. Value should be set to 1 or 0. |

1. Use cmake, type in Terminal:

$ make

1. Run the program

$ ./Indexer

# **how it works internally**

## **Main Components:**

1. ***DataLoader***: overall controller. Call rest components to setup, sort, build Index of dataset, write Inverted Index Structure, Lexicon Structure, Page (docID-to-URL) Table into files.
2. ***DocTable***: page table class.
3. ***InvertedIndex***: Inverted Index Structure class.
4. ***Lexicon***: Lexicon Structure class.

## **Workflow:**

1. **Index Setup**

*DataLoader*call *read\_all* function to read compressed dataset and generate postings.

1. The data will be read into buffer of size 50kb. The program will parse each Document’s docNO, total number of words, each word and its corresponding frequency according to ‘.trec’ format. The program will assign a docID automatically according to the order the documents are parsed.
2. For each distinct word in a document, calculate its frequency. Get a *SortedPosting List* and insert each posting into *InvertedIndex* one by one.
3. When the *InvertedIndex* size bigger than *FILE\_INDEX\_CHUNK*, write *InvertedIndex* in a bit/text format file in *INDEX\_FILE\_FOLDER\_PATH*, then clear the *InvertedIndex*. After parsing all documents, write remaining *InvertedIndex* in file.
4. After parsing all documents, *DocTable* write Page Table to *DOC\_TABLE\_PATH*.
5. **Disks and I/O efficient Sorting**

*DataLoader*call *mergeIndexToOne* function, using **merge-sort** algorithm to merge files in *INDEX\_FILE\_FOLDER\_PATH*.

1. Because docIDs are assigned according to the order the documents are parsed and postings in *InvertedIndex are sorted*, words in *InvertedIndex* List and docIDs of corresponding items in each unmerged file in *INDEX\_FILE\_FOLDER\_PATH* are in ascending order.
2. If the number of unmerged files is greater than 2, merge two of them in order and write the sorted files to a new file under *INDEX\_FILE\_FOLDER\_PATH.* The two original files are marked as merged, the new generated file is marked as unmerged.

Details about merging *file1* and *file2* to *dst\_file*:

1. Pointer *p1* and *p2* point to reading position of file1 and file2 respectively. Get Inverted Index (e.g. *wordi: {begin\_pointeri, end\_pointeri, docID\_numsi}*) *index1, index2* in *file1* and *file2* respectively.
2. Get *word1*, *word2* of *file1* and *file2* respectively. If *word1<word2*,

write *index1* in *dst\_file*. If *word1>word2*, write *index2* in *dst\_file*.

If *word1=word2*, merge *index1* and *index2* in *dst\_file*.

For merging *index1* and *index2,* compare the first docID of *index1* and *index2*, the items with smaller docID should be put first.

1. Repeat II until *p1* of *file1* or *p2* of *file 2* move to the end of corresponding file.
2. Write the remaining *file1* or *file2* to *dst\_file.*
3. Repeat b. until unmerged files no greater than 2. Those 2 unmerged files will be merged in Index Building part.
4. **Index Building**

*Lexicon* call *Build* function to build Index. Final Index will be built during remaining two unmerged file merging process. Merging process is same in Disks and I/O efficient Sorting. Final Inverted Index will be written to *FINAL\_INDEX\_PATH.*

During merging, use varbyte algorithm to encode docID and frequency. *Lexicon* records begin pointer position, end pointer position in final inverted index file and number of docID of each word.

After Index Building, Lexicon Structure *is* written to *LEXICON\_PATH.*

# **how long it takes on the provided data set**

Build Postings and TEMP Inverted Index Need 11835.2s

Merge Inverted Index Need 4689.11s

Write Lexicon Structure Need 65.8997s

Total time is 123260.21s

Total time is about 4.61h. If use unix sort to build temporary Inverted Index, total time is about 9793.52s(2.72h).

# **how large the resulting index files are**

After compress, the resulting index is 5.96 GB in bin format. Before compress, the resulting index is 13.62 GB in ASCII format.