Overview

This project consists of 3 components

- trec reader
  - trec_reader.cpp
  - trec_reader.h
- index builder
  - index_builder.cpp
  - index_builder.h
- data compressor
  - data_compress.cpp
  - data_compress.h
- main function
  - main.cpp
  - thread_pool.h

trec reader

- read trec file in block of 200,000 bytes (or 200KB) at a time
  - if too small, the read operation from disk will be too frequent
  - if too large, parsing the input string will take too much time
- parse and separate documents using <DOC></DOC> and other xml elements
- assign document id for index building
- save meta data of these documents
  - document id (used for index building)
  - document number/title (original identifier)
  - document url
  - the positions/offsets of these documents (for fast retrieval in the future)
  - the lengths of these documents (for fast retrieval in the future)

index builder

- accept documents parsed by the trec reader
- calculate the appearance of each word in the document and build inverted index based on it
- write temporary file to release memory
  - *temp<worker_id>-<file_id>*
- merge temporary file into final index output file
  - *index<worker_id>.out*

- save meta data of these index
  - the positions/offsets of the postings lists (for fast retrieval in the future)
  - the lengths of these postings lists (for fast retrieval in the future)
  - the start document id of the postings lists
  - the end document id of the postings lists
  - the number of documents contained in the postings lists

data compressor

- compress/decompress lexicon/word
  - take two (previous and current) string and encode the current based on the previous, for instance

    ```
    compressText("apple", "apply") => "4:y"
    ```

- compress/decompress a squence of numbers
  - if sorted, take the differences first
  - use varbyte encoding turn a list of numbers into string (a squence of bytes)

main function

- parse arguments to configure the builder(s)

```
~$ ./main -h
Usage: ./main [-m <maxmem>] [-c <compress>] [-w <workers>]
-m | --mem <maxmem>
        Numeric value, could have case-insensitive suffix k, m, g with no space.

        This option indicates the cache size for posting lists before writing into temporary file,
        On the other hand, it indicates the size of the temporary files.
        At least 256 (bytes), default is 2MB.
-c | --compress <compress>
        Boolean value, possible values are 1, 0, true, false.

        This option indicates if the data compression is applied. (Disable for easier debugging).
        Default is true.
-w | --worker <workers>
        Numeric value, possible values are 1, 0, true, false.

        This option determines the number of workers/threads to be used.
        Each worker (index builder) generate an result file (index.out), so it can be used to determine
        the size of the result file (index<worker-id>.out), basically each of which would be similar size.
        Default is 3.
```

- allocate the documents equally to the index builders using thread pool designed in *thread_pool.h*

Additional components in the projects

These parts of code is written before the trec file dataset determined to be used.

HTTP Request Sender / HTTP Formater/Convertor / XML parser

- contained in
    - *utility.cpp*
    - *utility.h*
    - *tinyxml2.cpp* (downloaded)
    - *tinyxml2.h* (downloaded)
- send http request to specified url and download the HTML pages if existed
- format and cleanup HTML tags
- convert HTML to XML
- extract content from XML

Workflow

- main() calls readdoc() from TrecReader object to obtain document text
- then enqueue() the parsing task into thread pool
    - objects of IndexBuilder keep taking the job from the queue
    - an object parses a document at a time
        - even there are idle threads (threads taking jobs using the same index builder, in order to make sure that the document id is arranged in ascending order in temp/result file
    - put word counter into temp file if the cache reaches the memory limits
        - practically speaking, when the counter size reaches maxmem / sizeof(posting)
    - merge all the temp files it generated (labelled by its worker id) into a result file
        - read *temp files* blocks by blocks as input buffer inbuff[i]
            - the default setting is 200 lines
        - take all the elements at the front and build a priority queue from them
            - the min heap is based on the lexicon/word
        - merge the postings lists that share the same lexicon/word
        - then merge the postings that share the same docIDs
        - compress lexicon/word and the sequences of docIDs and frequencies if compressing arguments is set to true or left out
        - append this postings list into *result file*

Usage

```
make
make init
# for argument detailed info
./main --help
# example
./main -w 10
```

There is a subset of the real 23GB file contained only 10 documents for testing and debugging, which is also the default choice. If you would preferred self-prepared dataset for testing, modified it at 107th line in main.cpp.

Performance

The program takes about an hour to read, separate, and enqueue the whole (23GB) file. Then about 15 hours to build the inverted index completely with 3 workers/threads.