National University of Sciences and Technology

Department of Computing

Data Structures and Algorithms (Code Submission)

Forward And Inverted Index

Group Members

Name	CMS ID	Group
Aftab Akhtar	244301	A
Usama Ahmed Siddiquie	241886	B
Muhammad Ali	266079	C

Demo Instructions

You need NLTK installed for this demo. The stopwords corpus and punkt for PorterStemmer is required.

To demo the features required by assignment 2, go to the **run.py** file. This is the file from which all other modules are called. There are 3 commented lines of code here. Uncomment each line one at a time and run. Each operation also prints some data to show what has been generated.

If the index is already generated, it will be overwritten. To prevent this from happening every time we add a new documents, you have to create a new batch in the dataset folder and specify in the generator modules to just look at those. Then the indexing will only be done for those new documents.

1. **Generating Lexicon:**

   The lexicon is generated. The lexicon is stored in a dictionary so there is no need to sort it as search is done in constant time.

2. **Generating Forward Index**

   The next line generates the forward index using multiple threads. We made a slightly different design decision here compared to the original Google paper. Rather than creating buckets based on wordIDs and having duplications we created buckets based on docIDs which eliminates duplicates. This does make inverting them less convenient but saves space as the forward index is less useful then the inverted index during search.

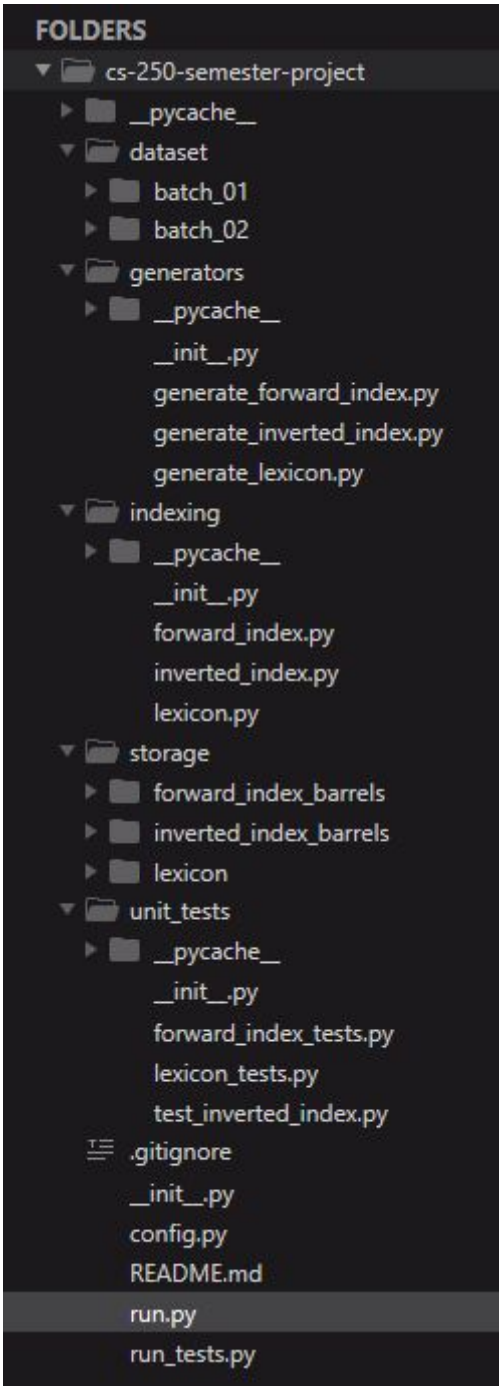
3. **Generating Inverted Index**

   Inverted indexes are generated by multiple threads and hence we need to have to first generated temporary indexes and then merge them. For a large number of large files this approach is faster. It does require some extra space but the temporary files are deleted as soon as they are merged into the main index. So for a large number of threads, this is not an issue. We used 2 threads in the demo.

   We have also chosen not to save the hit list with the inverted index as we can get that information during the search ranking from the forward index in constant time as well and save space.

Project Structure

The structure of this project can be seen in the picture attached. Different directories and their description is given below:



dataset: contains the dataset divided into batches.

indexing: contain code for forward/inverted index as well as for lexicon.

storage: contain the forward/inverted barrels and lexicon stored as binary.

unit_test: contain code for testing different functionality of project.

generator: will contain code for generating forward/inverted index using multiple threads.

Testing Functionality

To test the functionality of different modules in the project first understand the following files / scripts:

run_test.py: is the file through which we can test the functionality of other modules (lexicon, forward/inverted index). To test a specific functionality, we will first make changes as required to its respective unit_test/* file and then we will import and call the functions to be checked in the **run.py**. After that, we can simply compile and run the run.py file.

config.py: setup all the paths for different modules (**already setup**)

unit_tests/lexicon_tests: is setup for testing different functions defined in lexicon.py file.

unit_tests/forward_index_tests: is setup for testing functionality of forward_index.py.

unit_tests/test_inverted_index: is setup for testing inverted_index.py

FEATURES

Lexicon: The documents are converted to word_id's and dataset is converted into a lexicon.

Forward_Index: Documents are transformed as:

```
{
    doc_id: {
        word_id : [appearances]
        .
        .
    }
}
```

Inverted_Index: Documents are transformed as:

```
{
    word_id: [doci_id, doc_id, doc_id]
    .
    .
}
```