Search Engine Project - DSA

A search engine roughly based on the paper *The Anatomy of a Large-Scale Hypertextual Web Search Engine* by Sergey Brin and Lawrence Page

Group Members

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

Using this Repository

Dependencies

Type the following command in Terminal to download required packages

```
pip install -r requirements.txt
```

Then you need to download the following for NLTK:

1. Stopwords Corpus
2. Punkt Tokenizer Model

Type this in the python shell to get these

```
import nltk
nltk.download()
```

Demo

Live Server Demo

To get a demo in the browser for the search engine, run the run_server.py file and go to the url provided in the CLI.

CLI Demo (Detailed)

To demo the workings of this project run run.py in the terminal. User the following commands to check different aspects of the project

1. Lexicon creation/updating

    ```
    run.py generate_lexicon --b_range=a,b --d=1
    // a and b are the number of the starting and ending batch to update/create lexicon from.
    // --d set to 1 prints some demo results
    ```

2. Forward Index creation/update

```
run.py generate_forward_index --b_range=a,b --d=1
// a and b are the number of the starting and ending batch to update/create forward index.
// --d set to 1 prints some demo results
```

3. Inverted Index creation/update

```
run.py generate_inverted_index --b=batch_00a,batch_00b --d=1
// a,b... are numbers of forward index batches to create inverted index from
// --d set to 1 prints some demo results
```

4. Search

```
run.py search --q='query'
// query is the text to search
```

Working

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.

4. Search: Search works by tokenizing the search query and then looking in the inverted index to find all relevant documents. These documents are ranked by the number of hits and the proximity of those hits as they appear in the query.

Unit Tests

Proper unit tests have only been written for inverted index so far

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

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

unit_tests/test_inverted_index: is setup for testing inverted_index.py